Greedy Algorithms

Antonio Carzaniga

Faculty of Informatics
Università della Svizzera italiana

May 24, 2022

Outline

- Greedy strategy
- Examples
- Activity selection
- Huffman coding

- Find the MST of $G = (V, E)$ with $w : E \longrightarrow \mathbb{R}$
  - find a $T \subseteq E$ that is a *minimum-weight spanning tree*

Recap on MST Algorithms

- Find the MST of $G = (V, E)$ with $w : E \longrightarrow \mathbb{R}$
  - find a $T \subseteq E$ that is a *minimum-weight spanning tree*

- We naturally decompose the problem in a series of choices

- Find the MST of $G = (V, E)$ with $w : E \longrightarrow \mathbb{R}$
  - find a $T \subseteq E$ that is a *minimum-weight spanning tree*
- We naturally decompose the problem in a series of choices
  - at each point we have a partial solution $A \subseteq T$

Recap on MST Algorithms

- Find the MST of $G = (V, E)$ with $w : E \longrightarrow \mathbb{R}$
  - find a $T \subseteq E$ that is a *minimum-weight spanning tree*

- We naturally decompose the problem in a series of choices
  - at each point we have a partial solution $A \subseteq T$
  - we have a number of choices on how to extend A

Recap on MST Algorithms

- Find the MST of $G = (V, E)$ with $w : E \longrightarrow \mathbb{R}$
  - find a $T \subseteq E$ that is a *minimum-weight spanning tree*

- We naturally decompose the problem in a series of choices
  - at each point we have a partial solution $A \subseteq T$
  - we have a number of choices on how to extend A
  - we make a “greedy” choice by selecting the *lightest* edge that does not violate the constraints of the MST problem

- Find the MST of $G = (V, E)$ with $w : E \longrightarrow \mathbb{R}$
  - find a $T \subseteq E$ that is a *minimum-weight spanning tree*

- We naturally decompose the problem in a series of choices
  - at each point we have a partial solution $A \subseteq T$
  - we have a number of choices on how to extend A
  - we make a "greedy" choice by selecting the *lightest* edge that does not violate the constraints of the MST problem

```
GENERIC-MST(G, w)
A = ∅
while A is not a spanning tree
    find a safe edge e = (u, v) // the lightest that...
    A = A ∪ {e}
```

Designing a Greedy Algorithm

Designing a Greedy Algorithm

1. Cast the problem as one where
    - we make a ***greedy choice***, and
    - we are left with a ***subproblem***

Designing a Greedy Algorithm

1. Cast the problem as one where
   - we make a ***greedy choice***, and
   - we are left with a ***subproblem***
2. Prove that there is always a solution to the original problem that contains the greedy choice we make
   - i.e., that the greedy choice always leads to an optimal solution
     - not necessarily always the same one

Designing a Greedy Algorithm

1. Cast the problem as one where
   - we make a ***greedy choice***, and
   - we are left with a ***subproblem***
2. Prove that there is always a solution to the original problem that contains the greedy choice we make
   - i.e., that the greedy choice always leads to an optimal solution
     - not necessarily always the same one
3. Prove that the remaining subproblem is such that
   - combining the greedy choice with the optimal solution of the subproblem gives an optimal solution to the original problem

The Greedy-Choice Property

- The first key ingredient of a greedy strategy is the following

  greedy-choice property: *one can always arrive at a globally optimal solution by making a locally optimal choice*

The Greedy-Choice Property

- The first key ingredient of a greedy strategy is the following

  greedy-choice property: *one can always arrive at a globally optimal solution by making a locally optimal choice*

- At every step, we consider only what is best in the current problem
  - not considering the results of the subproblems

Optimal Substructure

- The second key ingredient of a greedy strategy is the following

  optimal-substructure property: *an optimal solution to the problem contains within it optimal solutions to subproblems*

Optimal Substructure

- The second key ingredient of a greedy strategy is the following

  optimal-substructure property: *an optimal solution to the problem contains within it optimal solutions to subproblems*

- It is natural to prove this by induction
  - if the solution to the subproblem is optimal, then combining the greedy choice with that solution yields an optimal solution

Example

- The absolutely trivial *gift-selection problem*

- The absolutely trivial *gift-selection problem*
  - out of a set $X = \{x_1, x_2, \ldots, x_n\}$ of valuable objects, where $v(x_i)$ is the value of x_i

- The absolutely trivial *gift-selection problem*
  - out of a set $X = \{x_1, x_2, \ldots, x_n\}$ of valuable objects, where $v(x_i)$ is the value of x_i
  - you will be given, as a gift, k objects of your choice

- The absolutely trivial *gift-selection problem*
  - out of a set $X = \{x_1, x_2, \ldots, x_n\}$ of valuable objects, where $v(x_i)$ is the value of x_i
  - you will be given, as a gift, k objects of your choice
  - how do you maximize the total value of your gifts?

- The absolutely trivial *gift-selection problem*
  - out of a set $X = \{x_1, x_2, \ldots, x_n\}$ of valuable objects, where $v(x_i)$ is the value of x_i
  - you will be given, as a gift, k objects of your choice
  - how do you maximize the total value of your gifts?

- *Decomposition:* choice plus subproblem

- The absolutely trivial *gift-selection problem*
  - out of a set $X = \{x_1, x_2, \ldots, x_n\}$ of valuable objects, where $v(x_i)$ is the value of x_i
  - you will be given, as a gift, k objects of your choice
  - how do you maximize the total value of your gifts?

- *Decomposition:* choice plus subproblem
  - ***greedy choice:*** pick x_i such that $v(x_i) = \max_{x \in X} v(x)$
  - ***subproblem:*** $X' = X - \{x_i\}$, $k' = k - 1$ (same value function v)

- The absolutely trivial *gift-selection problem*
  - out of a set $X = \{x_1, x_2, \ldots, x_n\}$ of valuable objects, where $v(x_i)$ is the value of x_i
  - you will be given, as a gift, k objects of your choice
  - how do you maximize the total value of your gifts?
- *Decomposition:* choice plus subproblem
  - ***greedy choice:*** pick x_i such that $v(x_i) = \max_{x \in X} v(x)$
  - ***subproblem:*** $X' = X - \{x_i\}$, $k' = k - 1$ (same value function v)
- *Greedy-choice property*

Example

- The absolutely trivial *gift-selection problem*
  - out of a set $X = \{x_1, x_2, \ldots, x_n\}$ of valuable objects, where $v(x_i)$ is the value of x_i
  - you will be given, as a gift, k objects of your choice
  - how do you maximize the total value of your gifts?
- *Decomposition:* choice plus subproblem
  - ***greedy choice:*** pick x_i such that $v(x_i) = \max_{x \in X} v(x)$
  - ***subproblem:*** $X' = X - \{x_i\}$, $k' = k - 1$ (same value function v)
- *Greedy-choice property*
  - if $v(x_i) = \max_{x \in X} v(x)$, then there is a globally optimal solution A that contains x_i

Example

- The absolutely trivial *gift-selection problem*
  - out of a set $X = \{x_1, x_2, \ldots, x_n\}$ of valuable objects, where $v(x_i)$ is the value of x_i
  - you will be given, as a gift, k objects of your choice
  - how do you maximize the total value of your gifts?
- *Decomposition:* choice plus subproblem
  - ***greedy choice:*** pick x_i such that $v(x_i) = \max_{x \in X} v(x)$
  - ***subproblem:*** $X' = X - \{x_i\}$, $k' = k - 1$ (same value function v)
- *Greedy-choice property*
  - if $v(x_i) = \max_{x \in X} v(x)$, then there is a globally optimal solution A that contains x_i
- *Optimal-substructure property*

Example

- The absolutely trivial *gift-selection problem*
  - out of a set $X = \{x_1, x_2, \ldots, x_n\}$ of valuable objects, where $v(x_i)$ is the value of x_i
  - you will be given, as a gift, k objects of your choice
  - how do you maximize the total value of your gifts?

- *Decomposition:* choice plus subproblem
  - ***greedy choice:*** pick x_i such that $v(x_i) = \max_{x \in X} v(x)$
  - ***subproblem:*** $X' = X - \{x_i\}$, $k' = k - 1$ (same value function v)

- *Greedy-choice property*
  - if $v(x_i) = \max_{x \in X} v(x)$, then there is a globally optimal solution A that contains x_i

- *Optimal-substructure property*
  - if $v(x_i) = \max_{x \in X} v(x)$ and A' is an optimal solution for $X' = X - \{x_i\}$, then $A' \subset A$

Observation

- *Inventing* a greedy algorithm is easy
  - it is easy to come up with greedy choices

Observation

- *Inventing* a greedy algorithm is easy
  - it is easy to come up with greedy choices

- *Proving it optimal* may be difficult

  - requires deep understanding of the ***structure of the problem***

Making Change

- My favorite pasta lunch typically costs Fr. 15.20; I usually pay with a Fr. 20 bill, and get Fr. 4.80 of change

- My favorite pasta lunch typically costs Fr. 15.20; I usually pay with a Fr. 20 bill, and get Fr. 4.80 of change

  Question: how can I get the least amount of coins?

  (Available denominations: 5, 2, 1, 0.5, 0.2, 0.1)

- My favorite pasta lunch typically costs Fr. 15.20; I usually pay with a Fr. 20 bill, and get Fr. 4.80 of change

  Question: how can I get the least amount of coins?

  (Available denominations: 5, 2, 1, 0.5, 0.2, 0.1)

  Solution: $2 \times 2 + 0.5 + 0.2 + 0.1 = 4.8$ (5 coins/bills)

- My favorite pasta lunch typically costs Fr. 15.20; I usually pay with a Fr. 20 bill, and get Fr. 4.80 of change

  Question: how can I get the least amount of coins?

  (Available denominations: 5, 2, 1, 0.5, 0.2, 0.1)

  Solution: $2 \times 2 + 0.5 + 0.2 + 0.1 = 4.8$ (5 coins/bills)

- Is this a greedy problem?

- My favorite pasta lunch typically costs Fr. 15.20; I usually pay with a Fr. 20 bill, and get Fr. 4.80 of change

  Question: how can I get the least amount of coins?

  (Available denominations: 5, 2, 1, 0.5, 0.2, 0.1)

  Solution: $2 \times 2 + 0.5 + 0.2 + 0.1 = 4.8$ (5 coins/bills)

- Is this a greedy problem?

- Suppose you are in the US and need to make $4.80 of change; available denominations are $5, $1, $0.25, $0.1, and $.01 (you are out of “nickels”)

Making Change

- My favorite pasta lunch typically costs Fr. 15.20; I usually pay with a Fr. 20 bill, and get Fr. 4.80 of change

  Question: how can I get the least amount of coins?

  (Available denominations: 5, 2, 1, 0.5, 0.2, 0.1)

  Solution: $2 \times 2 + 0.5 + 0.2 + 0.1 = 4.8$ (5 coins/bills)

- Is this a greedy problem?

- Suppose you are in the US and need to make \$4.80 of change; available denominations are \$5, \$1, \$0.25, \$0.1, and \$.01 (you are out of "nickels")

  Greedy: $4 \times 1 + 3 \times 0.25 + 5 \times 0.01 = 4.8$ (12 coins/bills)

- My favorite pasta lunch typically costs Fr. 15.20; I usually pay with a Fr. 20 bill, and get Fr. 4.80 of change

  Question: how can I get the least amount of coins?

  (Available denominations: 5, 2, 1, 0.5, 0.2, 0.1)

  Solution: $2 \times 2 + 0.5 + 0.2 + 0.1 = 4.8$ (5 coins/bills)

- Is this a greedy problem?

- Suppose you are in the US and need to make $4.80 of change; available denominations are $5, $1, $0.25, $0.1, and $.01 (you are out of "nickels")

  Greedy: $4 \times 1 + 3 \times 0.25 + 5 \times 0.01 = 4.8$ (12 coins/bills)

  Optimal: $4 \times 1 + 2 \times 0.25 + 3 \times 0.1 = 4.8$ (9 coins/bills)

- A thief robbing a store finds n items
  - v_i is the value of item i
  - w_i is the weight of item i
  - W is the maximum weight that the thief can carry

Problem: choose which items to take to maximize the total value of the robbery

- A thief robbing a store finds n items
  - v_i is the value of item i
  - w_i is the weight of item i
  - W is the maximum weight that the thief can carry

  Problem: choose which items to take to maximize the total value of the robbery

- Is this a greedy problem?

- A thief robbing a store finds n items
  - v_i is the value of item i
  - w_i is the weight of item i
  - W is the maximum weight that the thief can carry

  Problem: choose which items to take to maximize the total value of the robbery

- Is this a greedy problem?

- **Exercise:**
  1. formulate a reasonable greedy choice
  2. prove that it doesn't work with a counter-example
  3. go back to (1) and repeat a couple of times

Fractional Knapsack Problem

Fractional Knapsack Problem

- A thief robbing a store finds n items
  - v_i is the value of item i
  - w_i is the weight of item i
  - W is the maximum weight that the thief can carry
  - the thief may take any *fraction* of an item (with the corresponding proportional value)

Problem: choose which items, or fractions of items to take to maximize the total value of the robbery

Fractional Knapsack Problem

- A thief robbing a store finds n items
  - v_i is the value of item i
  - w_i is the weight of item i
  - W is the maximum weight that the thief can carry
  - the thief may take any *fraction* of an item (with the corresponding proportional value)

  Problem: choose which items, or fractions of items to take to maximize the total value of the robbery

- Is this a greedy problem?

Fractional Knapsack Problem

- A thief robbing a store finds n items
  - v_i is the value of item i
  - w_i is the weight of item i
  - W is the maximum weight that the thief can carry
  - the thief may take any *fraction* of an item (with the corresponding proportional value)

  Problem: choose which items, or fractions of items to take to maximize the total value of the robbery

- Is this a greedy problem?

- **Exercise:** prove that it is a greedy problem

Activity-Selection Problem

- A conference room is shared among different activities
  - $S = \{a_1, a_2, \ldots, a_n\}$ is the set of proposed activities
  - activity a_i has a *start time* s_i and a *finish time* f_i
  - activities a_i and a_j are *compatible* if either $f_i \leq s_j$ or $f_j \leq s_i$

Activity-Selection Problem

- A conference room is shared among different activities
  - $S = \{a_1, a_2, \ldots, a_n\}$ is the set of proposed activities
  - activity a_i has a *start time* s_i and a *finish time* f_i
  - activities a_i and a_j are *compatible* if either $f_i \leq s_j$ or $f_j \leq s_i$

Problem: find the largest set of compatible activities

- A conference room is shared among different activities
  - $S = \{a_1, a_2, \ldots, a_n\}$ is the set of proposed activities
  - activity a_i has a *start time* s_i and a *finish time* f_i
  - activities a_i and a_j are *compatible* if either $f_i \leq s_j$ or $f_j \leq s_i$

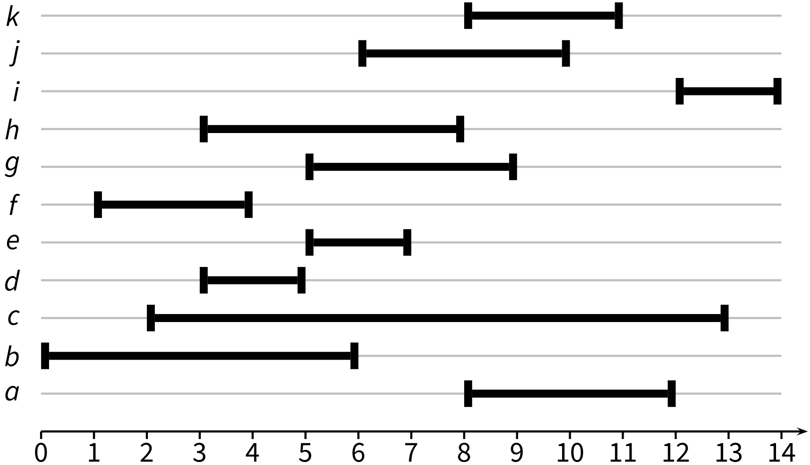
Problem: find the largest set of compatible activities

- Example

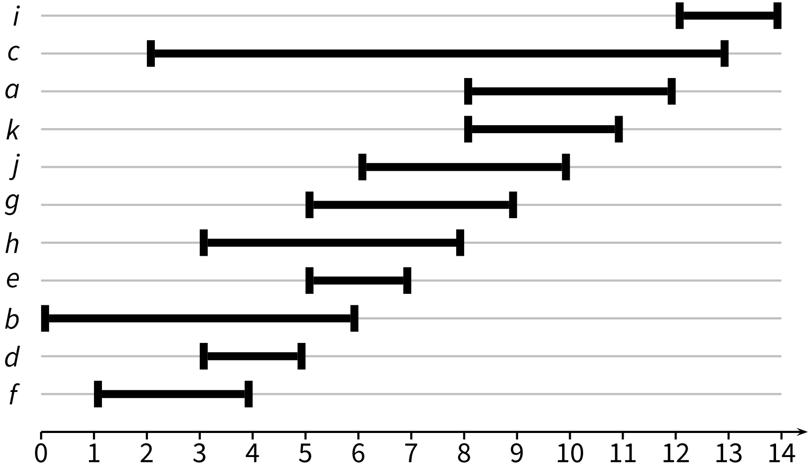
activity	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*	*k*
start	8	0	2	3	5	1	5	3	12	6	8
finish	12	6	13	5	7	4	9	8	14	10	11

- Is there a greedy solution for this problem?

Activity-Selection Problem (2)

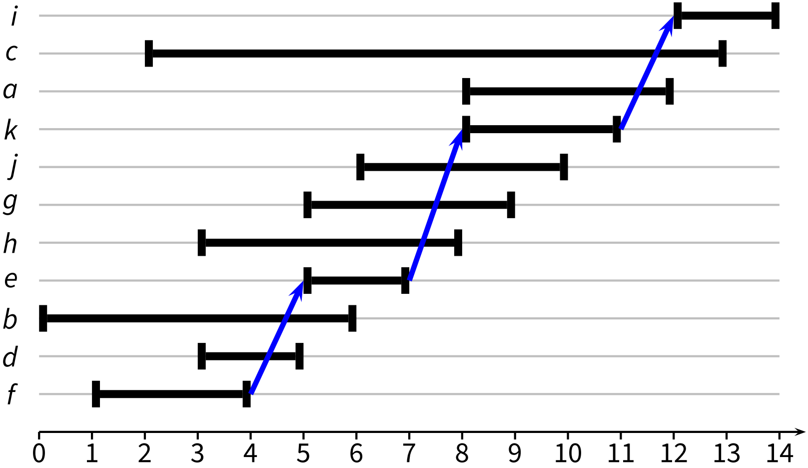


Activity-Selection Problem (3)



Greedy choice: ***earliest finish time***

Activity-Selection Problem (3)



Greedy choice: ***earliest finish time***

Activity Selection is a Greedy Problem

Activity Selection is a Greedy Problem

- ***Greedy choice:*** take $a_x \in S$ s.t. $f_x \leq f_i$ for all $a_i \in S$

Activity Selection is a Greedy Problem

- ***Greedy choice:*** take $a_x \in S$ s.t. $f_x \leq f_i$ for all $a_i \in S$

  Prove: there is an optimal solution OPT^* that contains a_x

Activity Selection is a Greedy Problem

- ***Greedy choice:*** take $a_x \in S$ s.t. $f_x \leq f_i$ for all $a_i \in S$

  Prove: there is an optimal solution OPT^* that contains a_x

  Proof: (by contradiction)

  - assume $a_x \notin OPT$

Activity Selection is a Greedy Problem

- ***Greedy choice:*** take $a_x \in S$ s.t. $f_x \leq f_i$ for all $a_i \in S$

  Prove: there is an optimal solution OPT^* that contains a_x

  Proof: (by contradiction)

  - assume $a_x \notin OPT$
  - let $a_m \in OPT$ be the earliest-finish activity in OPT

Activity Selection is a Greedy Problem

- ***Greedy choice:*** take $a_x \in S$ s.t. $f_x \leq f_i$ for all $a_i \in S$

  Prove: there is an optimal solution OPT^* that contains a_x

  Proof: (by contradiction)

  - assume $a_x \notin OPT$
  - let $a_m \in OPT$ be the earliest-finish activity in OPT
  - construct $OPT^* = OPT \setminus \{a_m\} \cup \{a_x\}$

Activity Selection is a Greedy Problem

- ***Greedy choice:*** take $a_x \in S$ s.t. $f_x \leq f_i$ for all $a_i \in S$

  Prove: there is an optimal solution OPT^* that contains a_x

  Proof: (by contradiction)

  - assume $a_x \notin OPT$
  - let $a_m \in OPT$ be the earliest-finish activity in OPT
  - construct $OPT^* = OPT \setminus \{a_m\} \cup \{a_x\}$
  - OPT^* is *valid*
    Proof:
    - every activity $a_i \in OPT \setminus \{a_m\}$ has a starting time $s_i \geq f_m$, because a_m is compatible with a_i (so either $f_i < s_m$ or $s_i > f_m$) and $f_i > f_m$, because a_m is the earliest-finish activity in OPT
    - therefore, every activity a_i is compatible with a_x, because $s_i \geq f_m \geq f_x$

Activity Selection is a Greedy Problem

- ***Greedy choice:*** take $a_x \in S$ s.t. $f_x \leq f_i$ for all $a_i \in S$

  Prove: there is an optimal solution OPT^* that contains a_x

  Proof: (by contradiction)

  - assume $a_x \notin OPT$
  - let $a_m \in OPT$ be the earliest-finish activity in OPT
  - construct $OPT^* = OPT \setminus \{a_m\} \cup \{a_x\}$
  - OPT^* is *valid*
    Proof:
    - every activity $a_i \in OPT \setminus \{a_m\}$ has a starting time $s_i \geq f_m$, because a_m is compatible with a_i (so either $f_i < s_m$ or $s_i > f_m$) and $f_i > f_m$, because a_m is the earliest-finish activity in OPT
    - therefore, every activity a_i is compatible with a_x, because $s_i \geq f_m \geq f_x$
  - thus OPT^* is an *optimal* solution, because $|OPT^*| = |OPT|$

Activity Selection is a Greedy Problem (2)

Activity Selection is a Greedy Problem (2)

- ***Optimal-substructure property:*** having chosen a_x, let $S' \subset S$ be the set of activities compatible with a_x, that is, $S' = \{a_i \mid s_i \geq f_x\}$

Activity Selection is a Greedy Problem (2)

- ***Optimal-substructure property:*** having chosen a_x, let $S' \subset S$ be the set of activities compatible with a_x, that is, $S' = \{a_i \mid s_i \geq f_x\}$

  Prove: $OPT^* = \{a_x\} \cup OPT'$ is optimal for S if OPT' is optimal for S'

Activity Selection is a Greedy Problem (2)

- ***Optimal-substructure property:*** having chosen a_x, let $S' \subset S$ be the set of activities compatible with a_x, that is, $S' = \{a_i \mid s_i \geq f_x\}$

  Prove: $OPT^* = \{a_x\} \cup OPT'$ is optimal for S if OPT' is optimal for S'

  Proof: (by contradiction)

  - assume to the contrary that $|OPT^*| < |OPT|$, and therefore $|OPT'| < |OPT| - 1$

Activity Selection is a Greedy Problem (2)

- ***Optimal-substructure property:*** having chosen a_x, let $S' \subset S$ be the set of activities compatible with a_x, that is, $S' = \{a_i \mid s_i \geq f_x\}$

  Prove: $OPT^* = \{a_x\} \cup OPT'$ is optimal for S if OPT' is optimal for S'

  Proof: (by contradiction)

  - assume to the contrary that $|OPT^*| < |OPT|$, and therefore $|OPT'| < |OPT| - 1$
  - let a_m be the earliest-finish activity in OPT, and let $\overline{S} = \{a_i \mid s_i \geq f_m\}$

Activity Selection is a Greedy Problem (2)

- ***Optimal-substructure property:*** having chosen a_x, let $S' \subset S$ be the set of activities compatible with a_x, that is, $S' = \{a_i \mid s_i \geq f_x\}$

  Prove: $OPT^* = \{a_x\} \cup OPT'$ is optimal for S if OPT' is optimal for S'

  Proof: (by contradiction)

  - assume to the contrary that $|OPT^*| < |OPT|$, and therefore $|OPT'| < |OPT| - 1$
  - let a_m be the earliest-finish activity in OPT, and let $\overline{S} = \{a_i \mid s_i \geq f_m\}$
  - by construction, $OPT \setminus \{a_m\}$ is a solution for $\overline{S}$

- ***Optimal-substructure property:*** having chosen a_x, let $S' \subset S$ be the set of activities compatible with a_x, that is, $S' = \{a_i \mid s_i \geq f_x\}$

  Prove: $OPT^* = \{a_x\} \cup OPT'$ is optimal for S if OPT' is optimal for S'

  Proof: (by contradiction)

  - assume to the contrary that $|OPT^*| < |OPT|$, and therefore $|OPT'| < |OPT| - 1$
  - let a_m be the earliest-finish activity in OPT, and let $\overline{S} = \{a_i \mid s_i \geq f_m\}$
  - by construction, $OPT \setminus \{a_m\}$ is a solution for $\overline{S}$
  - by construction, $\overline{S} \subseteq S'$, so $OPT \setminus \{a_m\}$ is a solution also for S'

Activity Selection is a Greedy Problem (2)

- ***Optimal-substructure property:*** having chosen a_x, let $S' \subset S$ be the set of activities compatible with a_x, that is, $S' = \{a_i \mid s_i \geq f_x\}$

  Prove: $OPT^* = \{a_x\} \cup OPT'$ is optimal for S if OPT' is optimal for S'

  Proof: (by contradiction)

  - assume to the contrary that $|OPT^*| < |OPT|$, and therefore $|OPT'| < |OPT| - 1$
  - let a_m be the earliest-finish activity in OPT, and let $\overline{S} = \{a_i \mid s_i \geq f_m\}$
  - by construction, $OPT \setminus \{a_m\}$ is a solution for $\overline{S}$
  - by construction, $\overline{S} \subseteq S'$, so $OPT \setminus \{a_m\}$ is a solution also for S'
  - which means that there is a solution S' of size $|OPT| - 1$, which contradicts the main assumption that $|OPT'| < |OPT| - 1$

- Suppose you have a large sequence S of the six characters: ‘a’, ‘b’, ‘c’, ‘d’, ‘e’, and ‘f’
  - e.g., $n = |S| = 10^9$
- What is the most efficient way to store that sequence?

- Suppose you have a large sequence S of the six characters: ‘a’, ‘b’, ‘c’, ‘d’, ‘e’, and ‘f’
  - e.g., $n = |S| = 10^9$
- What is the most efficient way to store that sequence?
- First approach: compact fixed-width encoding

- Suppose you have a large sequence S of the six characters: ‘a’, ‘b’, ‘c’, ‘d’, ‘e’, and ‘f’
  - e.g., $n = |S| = 10^9$
- What is the most efficient way to store that sequence?
- First approach: compact fixed-width encoding
  - 6 symbols require 3 bits per symbol

- Suppose you have a large sequence S of the six characters: ‘a’, ‘b’, ‘c’, ‘d’, ‘e’, and ‘f’
  - e.g., $n = |S| = 10^9$
- What is the most efficient way to store that sequence?
- First approach: compact fixed-width encoding
  - 6 symbols require 3 bits per symbol
  - $3 \times 10^9/8 = 3.75 \times 10^8$ (a bit less than 400Mb)

- Suppose you have a large sequence S of the six characters: ‘a’, ‘b’, ‘c’, ‘d’, ‘e’, and ‘f’
  - e.g., $n = |S| = 10^9$
- What is the most efficient way to store that sequence?
- First approach: compact fixed-width encoding
  - 6 symbols require 3 bits per symbol
  - $3 \times 10^9/8 = 3.75 \times 10^8$ (a bit less than 400Mb)
- Can we do better?

Huffman Coding (2)

Huffman Coding (2)

- Consider the following encoding table:

symbol	*code*
a	000
b	001
c	010
d	011
e	100
f	101

- Consider the following encoding table:

symbol	*code*
a	000
b	001
c	010
d	011
e	100
f	101

- *Observation:* the encoding of ‘e’ and ‘f’ is a bit redundant
  - the second bit does not help us in distinguishing ‘e’ from ‘f’
  - in other words, if the first (most significant) bit is 1, then the second bit gives us no information, so it can be removed

Idea

Idea

- Variable-length code

symbol	*code*
a	000
b	001
c	010
d	011
e	10
f	11

- Encoding and decoding are well-defined and unambiguous

- Variable-length code

symbol	*code*
a	000
b	001
c	010
d	011
e	10
f	11

- Encoding and decoding are well-defined and unambiguous

- How much space do we save?

Idea

- Variable-length code

symbol	*code*
a	000
b	001
c	010
d	011
e	10
f	11

- Encoding and decoding are well-defined and unambiguous
- How much space do we save?
  - *not knowing the frequency of 'e' and 'f'*, we can't tell exactly

Idea

- Variable-length code

symbol	*code*
a	000
b	001
c	010
d	011
e	10
f	11

- Encoding and decoding are well-defined and unambiguous

- How much space do we save?
  - *not knowing the frequency of 'e' and 'f'*, we can't tell exactly

- Given the frequencies $f_a, f_b, f_c, \ldots$ of all the symbols in S

$$M = 3n(f_a + f_b + f_c + f_d) + 2n(f_e + f_f)$$

Problem Definition

Problem Definition

- Given a set of symbols C and a frequency function $f : C \longrightarrow [0, 1]$
- Find a code $E : C \longrightarrow \{0, 1\}^*$ such that

- Given a set of symbols C and a frequency function $f : C \longrightarrow [0, 1]$
- Find a code $E : C \longrightarrow \{0, 1\}^*$ such that
- E is a *prefix code*
  - no codeword $E(c_1)$ is the prefix of another codeword $E(c_2)$

- Given a set of symbols C and a frequency function $f : C \longrightarrow [0, 1]$
- Find a code $E : C \longrightarrow \{0, 1\}^*$ such that
- E is a *prefix code*
  - no codeword $E(c_1)$ is the prefix of another codeword $E(c_2)$
- The average codeword size

$$B(S) = \sum_{c \in C} f(c)|E(c)|$$

is minimal

Problem Definition (2)

Problem Definition (2)

- $E : C \longrightarrow \{0, 1\}^*$ defines binary strings, so we can represent E as a binary tree T

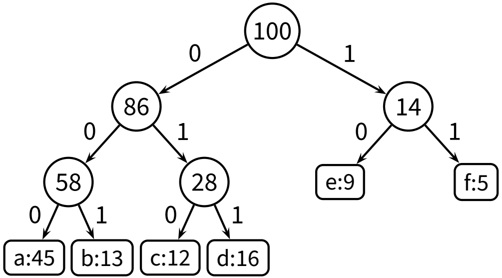
- $E : C \longrightarrow \{0, 1\}^*$ defines binary strings, so we can represent E as a binary tree T

sym.	*freq.*	*code*
a	45%	000
b	13%	001
c	12%	010
d	16%	011
e	9%	10
f	5%	11

Problem Definition (2)

- $E : C \longrightarrow \{0, 1\}^*$ defines binary strings, so we can represent E as a binary tree T

sym.	*freq.*	*code*
a	45%	000
b	13%	001
c	12%	010
d	16%	011
e	9%	10
f	5%	11

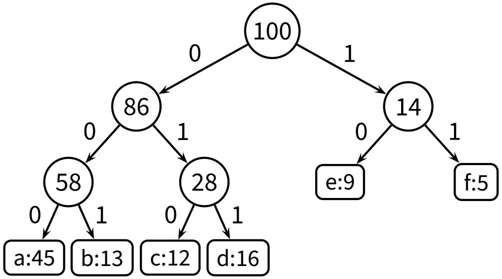


  - leaves represent symbols; internal nodes are prefixes
  - the code of a symbol c is the path from the root to c
  - the weight $f(v)$ of a node v is the frequency of its code/prefix

Problem Definition (2)

- $E : C \longrightarrow \{0, 1\}^*$ defines binary strings, so we can represent E as a binary tree T

sym.	*freq.*	*code*
a	45%	000
b	13%	001
c	12%	010
d	16%	011
e	9%	10
f	5%	11



- leaves represent symbols; internal nodes are prefixes
- the code of a symbol c is the path from the root to c
- the weight $f(v)$ of a node v is the frequency of its code/prefix

$$B(S) = n \sum_{c \in leaves(T)} f(c) depth(c) = n \sum_{v \in T} f(v)$$

Huffman Algorithm

```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
    create a new node z
    z.left = EXTRACT-MIN(Q)
    z.right = EXTRACT-MIN(Q)
    f(z) = f(z.left) + f(z.right)
    INSERT(Q, z)
return EXTRACT-MIN(Q)
```

Huffman Algorithm

```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
    create a new node z
    z.left = EXTRACT-MIN(Q)
    z.right = EXTRACT-MIN(Q)
    f(z) = f(z.left) + f(z.right)
    INSERT(Q, z)
return EXTRACT-MIN(Q)
```

- We build the code bottom-up

Huffman Algorithm

```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
        create a new node z
        z.left = EXTRACT-MIN(Q)
        z.right = EXTRACT-MIN(Q)
        f(z) = f(z.left) + f(z.right)
        INSERT(Q, z)
return EXTRACT-MIN(Q)
```

- We build the code bottom-up
- Each time we make the “greedy” choice of merging the two least frequent nodes (symbols or prefixes)

Example

```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
        create a new node z
        z.left = EXTRACT-MIN(Q)
        z.right = EXTRACT-MIN(Q)
        f(z) = f(z.left) + f(z.right)
        INSERT(Q, z)
return EXTRACT-MIN(Q)
```

sym.	*freq.*	*code*
a	45%	
b	13%	
c	12%	
d	16%	
e	9%	
f	5%	

Example

```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
        create a new node z
        z.left = EXTRACT-MIN(Q)
        z.right = EXTRACT-MIN(Q)
        f(z) = f(z.left) + f(z.right)
        INSERT(Q, z)
return EXTRACT-MIN(Q)
```

sym.	*freq.*	*code*
a	45%	
b	13%	
c	12%	
d	16%	
e	9%	
f	5%	

Example

```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
        create a new node z
        z.left = EXTRACT-MIN(Q)
        z.right = EXTRACT-MIN(Q)
        f(z) = f(z.left) + f(z.right)
        INSERT(Q, z)
return EXTRACT-MIN(Q)
```

sym.	*freq.*	*code*
a	45%	
b	13%	
c	12%	
d	16%	
e	9%	
f	5%	

Example

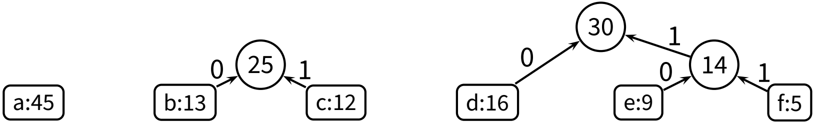
```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
        create a new node z
        z.left = EXTRACT-MIN(Q)
        z.right = EXTRACT-MIN(Q)
        f(z) = f(z.left) + f(z.right)
        INSERT(Q, z)
return EXTRACT-MIN(Q)
```

sym.	*freq.*	*code*
a	45%	
b	13%	
c	12%	
d	16%	
e	9%	
f	5%	

Example

```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
        create a new node z
        z.left = EXTRACT-MIN(Q)
        z.right = EXTRACT-MIN(Q)
        f(z) = f(z.left) + f(z.right)
        INSERT(Q, z)
return EXTRACT-MIN(Q)
```

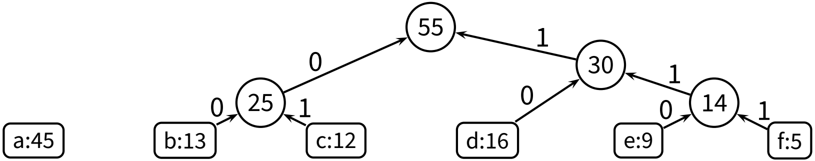
sym.	*freq.*	*code*
a	45%	
b	13%	
c	12%	
d	16%	
e	9%	
f	5%	



Example

```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
        create a new node z
        z.left = EXTRACT-MIN(Q)
        z.right = EXTRACT-MIN(Q)
        f(z) = f(z.left) + f(z.right)
        INSERT(Q, z)
return EXTRACT-MIN(Q)
```

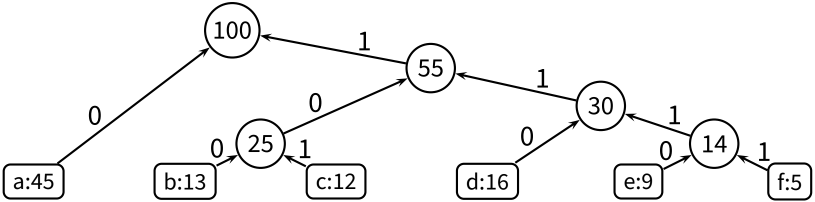
sym.	*freq.*	*code*
a	45%	
b	13%	
c	12%	
d	16%	
e	9%	
f	5%	



Example

```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
        create a new node z
        z.left = EXTRACT-MIN(Q)
        z.right = EXTRACT-MIN(Q)
        f(z) = f(z.left) + f(z.right)
        INSERT(Q, z)
return EXTRACT-MIN(Q)
```

sym.	*freq.*	*code*
a	45%	
b	13%	
c	12%	
d	16%	
e	9%	
f	5%	



Example

```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
        create a new node z
        z.left = EXTRACT-MIN(Q)
        z.right = EXTRACT-MIN(Q)
        f(z) = f(z.left) + f(z.right)
        INSERT(Q, z)
return EXTRACT-MIN(Q)
```

sym.	*freq.*	*code*
a	45%	0
b	13%	100
c	12%	101
d	16%	110
e	9%	1110
f	5%	1111