Greedy algorithms

Codes
Existence of printing codes
Shamon's trees
Graph (green of printing)

Whitmal spanning trees
Graph (green of printing)

Greedy algorithms

Greedy algorithms

Controber 20, 2017

Prints algorithm

October 20, 2017



Greedy algorithms • The problem: Suppose we have a text file made out of some finite all phabet $x_1, ..., x_n$ and we want to encode it to a binary code. Hufman Some finite all phabet $x_1, ..., x_n$ and we want to encode it to a binary code. It should look like: | It should look like: | Alimin spaning trees | Alimin | A

Greedy algorithms • The problem: Suppose we have a text file made out of some finite alphabet $x_1, ..., x_n$ and we want to encode it to a binary code. • It should look like: **Minimal abcdefgh** **One for the problem: Suppose we have a text file made out of some finite alphabet $x_1, ..., x_n$ and we want to encode it to a binary code. • It should look like: **Minimal abcdefgh** **One for the file such that: **And we would like to encode the file such that: **And we would like to encode the file such that: **And white the file such that: **And we would like to encode the file such that: **And we w

creedy algorithms • The problem: Suppose we have a text file made out of some finite alphabet x1,..., xn and we want to encode it to a binary code. • It should look like: • It should look like: • We would like to encode the file such that: • We would like to encode the file such that: • The prossible to reconstruct the original file from the encoded file (to decode the file). • The encoded file is of minimal length.

○ > C = (重) (重) (重) (□)

Encoding a text file - an example

Greedy algorithms Huffman codes Existence of

One possible encoding is: Codeword No. 01 10 11
 In this case, the length of the encoded file will be

$$\frac{N}{2} \cdot 2 + \frac{N}{4} \cdot 2 + \frac{N}{8} \cdot 2 + \frac{N}{8} \cdot 2 = 2N \text{ bits}$$

• Another option is: $\frac{\text{character}}{\text{Codeword}} \stackrel{A}{=} \frac{\text{B}}{10} \frac{\text{C}}{111}$ In this case, the length of the encoded file will be

$$\frac{N}{2} \cdot 1 + \frac{N}{4} \cdot 2 + \frac{N}{8} \cdot 3 + \frac{N}{8} \cdot 3 = 1.75N \text{ bits}$$

Vhich is better

Encoding a text file - an example

Greedy algorithms

• Suppose the file has N characters all of them from the set $\{A,B,C,D\}$ and we know that the frequencies in which they appear in the file are: $\frac{\text{character}}{\text{frequency}} \ \frac{A}{2} \ \frac{B}{4} \ \frac{C}{8} \ \frac{D}{8}$

 \bullet One possible encoding is: $\frac{\text{character}}{\text{Codeword}}$ $\frac{A}{00}$ $\frac{B}{01}$ $\frac{C}{10}$ $\frac{D}{11}$ In this case, the length of the encoded file will be

 $\frac{N}{2} \cdot 2 + \frac{N}{4} \cdot 2 + \frac{N}{8} \cdot 2 + \frac{N}{8} \cdot 2 = 2N \text{ bits}$

2 . 4 . 8 . 8

 $\frac{N}{2} \cdot 1 + \frac{N}{4} \cdot 2 + \frac{N}{8} \cdot 3 + \frac{N}{8} \cdot 3 = 1.75N \text{ bit.}$

/hich is better!

Encoding a text file - an example

Greedy algorithms

• Suppose the file has N characters all of them from the set $\{A,B,C,D\}$ and we know that the frequencies in which they appear in the file are: $\frac{\text{Character}}{\text{frequency}} \frac{A}{2} + \frac{B}{4} = \frac{D}{8}$

 \bullet One possible encoding is: $\frac{\text{character}}{\text{Codeword}}$ $\frac{\text{A}}{00}$ $\frac{\text{B}}{01}$ $\frac{\text{C}}{10}$ $\frac{11}{11}$ In this case, the length of the encoded file will be

 $\frac{N}{2}\cdot 2 + \frac{N}{4}\cdot 2 + \frac{N}{8}\cdot 2 + \frac{N}{8}\cdot 2 = 2N \text{ bits}$

• Another option is: $\frac{\text{character}}{\text{Codeword}} \frac{\text{A}}{\text{0}} \frac{\text{B}}{\text{10}} \frac{\text{C}}{\text{111}}$ In this case, the length of the encoded file will be

 $\frac{N}{2} \cdot 1 + \frac{N}{4} \cdot 2 + \frac{N}{8} \cdot 3 + \frac{N}{8} \cdot 3 = 1.75N \text{ bits}$

Which is better!

Greedy algorithms Codes Algorithms Codes Algorithms Codes Algorithms Algorithm Algo

Greedy algorithms Huffman codes file is to make sure that we can decode the encoded file is to make sure that no code word is the prefix (beginning) of another codeword. Definition: A prefix code is a collection of binary codewords $w_1, w_2, ..., w_n$ such that for any $i \neq j$ the codeword w_i is not the prefix of w_j . Prefix codes makes the decoding process especially simple. We will only study prefix codes. However this is not the only possible way to go.

○ > C = (重) (重) (重) (□)

Prefix codes

Greedy algorithms

• One way to make sure that we can decode the encoded

file is to make sure that no code word is the prefix

- Prefix codes makes the decoding process especially simple. We will only study prefix codes. However this is not the • Definition: A prefix code is a collection of binary codewords $w_1, w_2, ..., w_n$ such that for any $i \neq j$ the codeword w_i is not the prefix of w_i . (beginning) of another codeword. only possible way to go.

Prefix codes and binary trees

Greedy algorithms algorithms

Huffman codes

Existence of prefix codes
Shamon's Shamon's Huffman algorithm

- There is a one to one correspondence between prefix codes with m codewords and binary trees with m leaves.
- Example:
- We tag the edges going left by 0 and the edges going right by 1. Each leaf of the tree corresponds to a codeword according to the path from the root to the leaf.
- The collection of codewords that corresponds to the leaves of a binary tree is a prefix code. Moreover, any prefix code corresponds to some binary tree.

Greedy Greedy There is a one to one correspondence between prefix codes with *m* codewords and binary trees with *m* leaves. Huffman Example: Example: We tag the edges going left by 0 and the edges going right by 1. Each leaf of the tree corresponds to a codeword according to the path from the root to the leaf. The collection of codewords that corresponds to the leaves of a binary tree is a prefix code corresponds to some binary tree.

Greedy Greedy There is a one to one correspondence between prefix codes with *m* codewords and binary trees with *m* leaves. Huffman Example: Example: Opin the codes going left by 0 and the edges going right by 1. Each leaf of the tree corresponds to a codeword according to the path from the root to the leaf. The collection of codewords that corresponds to the leaves of a binary tree is a prefix code corresponds to some binary tree.

Greedy algorithms of Example: Statement Hufman codes Hufman codes Statement Hufman of Example: Statement Hufman of Example

• We tag the edges going left by 0 and the edges going right

by 1. Each leaf of the tree corresponds to a codeword

according to the path from the root to the leaf.

• The collection of codewords that corresponds to the leaves of a binary tree is a prefix code. Moreover, any prefix code

corresponds to some binary tree.

Greedy algorithms

 \bullet Clearly we can not find a prefix code for A,B and C with all lengths being 1.

If A=0 and B=1 we have no codeword for C

- We can ask the following question:
- Any guesses?
- The answer is on the next slide.

Greedy algorithms

- \bullet Clearly we can not find a prefix code for A,B and C with all lengths being 1. If A=0 and B=1 we have no codeword for C
- We can ask the following question: Given the lengthes $\{I_1,...,I_n\}$, when can we find a prefix code $\{w_1,...,w_n\}$ such that $|w_i|=I_i$?
- Any guesses?
- The answer is on the next slide.

Greedy algorithms

 \bullet Clearly we can not find a prefix code for A,B and C with all lengths being 1. If A=0 and B=1 we have no codeword for C

- We can ask the following question: Given the lengthes $\{l_1,...,l_n\}$, when can we find a prefix code $\{w_1,...,w_n\}$ such that $|w_i|=l_i$?
 - Any guesses ?
- The answer is on the next slide.

Greedy algorithms

 \bullet Clearly we can not find a prefix code for A,B and C with all lengths being 1.

If A=0 and B=1 we have no codeword for C

- We can ask the following question: Given the lengthes $\{I_1,...,I_n\}$, when can we find a prefix code $\{w_1,...,w_n\}$ such that $|w_i|=I_i$?
 - Any guesses?
- The answer is on the next slide.

Greedy algorithms

Existence of prefix codes Shannon's theorem Huffman

algorithm

Minimal

Spanning tree

Graphs (some

concepts and

definitions)

concepts and definitions)
Minimal spanning trees
Kruskal agorithm for MST

• Theorem: There exists a prefix code $\{w_1,...,w_n\}$ with lengths $I_1,...,I_n$ if and only if:

 $\sum_{i=1}^{i=n} 2^{-l_i} \le 1$

- Proof: By what we said about binary trees and prefix codes, it is enough to show that there exists a binary tree T with leaves $v_1, ..., v_n$ of heights $h_T(v_i) = l_i$ if and only if $\sum_{j=1}^{i=n} 2^{-l_j} \le 1$.
- Since the theorem is "if and only if" we will prove both directions.

Greedy algorithms

• Theorem: There exists a prefix code $\{w_1,...,w_n\}$ with lengths $l_1, ..., l_n$ if and only if:

 $\sum_{i=1}^{j=n} 2^{-l_i} \le 1$

T with leaves $v_1,...,v_n$ of heights $h_T(v_i)=l_i$ if and only if $\sum_{i=1}^{l=n}2^{-l_i}\leq 1.$ codes, it is enough to show that there exists a binary tree • Proof: By what we said about binary trees and prefix

• Since the theorem is "if and only if" we will prove both

>>

Greedy
algorithms
Huffman
coddes
Existence of
prefix codes
Shannon's
Huffman
Auffman
Auffman
Muffman

• Theorem: There exists a prefix code $\{w_1,...,w_n\}$ with lengths $I_1,...,I_n$ if and only if:

$$\sum_{i=1}^{i=n} 2^{-l_i} \le 1$$

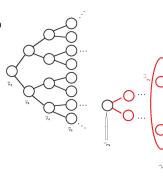
- <u>Proof:</u> By what we said about binary trees and prefix codes, it is enough to show that there exists a binary tree T with leaves $v_1,...,v_n$ of heights $h_T(v_i)=l_i$ if and only if $\sum_{i=1}^{l=n}2^{-l_i}\leq 1$.
 - Since the theorem is "if and only if" we will prove both directions.

Proof - first direction

Proot - Tirst directio

Greedy algorithms

• Assume there exists a binary tree T with leaves $v_1,...,v_n$ of heights $h_T(v_i)=I_i$. Denote $L=\max_{1\leq i\leq n}\{I_i\}$. If we complete each leaf of T as a full binary tree up to level L, we get a sub graph of the full binary tree of hight L, whose leaves are all of height L.



▲ □ ▼ ★ □ ▼ ■ ◆ ○ Q ○

Proof - first direction

Greedy algorithms

The sets of descendants of the different leaves are disjoint. The number of all the descendants of the original leaves is • A leaf in level l_i has 2^{L-l_i} descendants (in level L). no more than 2^L . Thus, we get:

$$2^L \ge \sum_{i=1}^n 2^{L-l_i} = 2^L \cdot \sum_{i=1}^n 2^{-l_i}$$
 This implies that $1 \ge \sum_{i=1}^n 2^{-l_i}$.

Proof - second direction

Greedy algorithms

On the other direction, assume that $1 \geq \sum_{i=1}^{n} 2^{-l_i}$. We prove by induction on n (the number of codewords) that there exists a binary tree with n leaves of heights $l_1, ..., l_n$. Assume, without loss of generality that $l_n = \max_i \{l_1, ..., l_n\}$. Since $1 \geq \sum_{i=1}^{n-1} 2^{-l_i}$, it follows from the induction hypothesis that there exists a binary tree T' with leaves $v_1, ..., v_{n-1}$ such that $h_{T'}(v_i) = l_i$.

If, like before, we complete all the leaves to full binary trees, up to level $L = I_n$, than we get $2^{I_n - I_i}$ leaves for each leaf in the original tree. It follows that:

$$\sum_{i=1}^{n-1} 2^{l_n - l_i} = 2^{l_n} \cdot \sum_{i=1}^{n-1} 2^{-l_i} \le 2^{l_n} (1 - 2^{-l_n}) = 2^{l_n} - 1$$

We see that at least one node in level I_n is not a descendant of $v_1, ..., v_{n-1}$. If we take T to be T' and that node (with all his ancestors) we get the desired tree.

Greedy algorithms

reflx code
$$w_1,...,w_n$$

$$\sum_{i=1}^n p_i |w_i|$$

- We denote the minimal value of $\sum_{i=1}^n p_i |w_i|$ by f(p). We define the entropy function to be: $H(p_1,...,p_n) = \sum_{i=1}^n p_i \log_2 \frac{1}{p_i}$ • Shannon's theorem: We have: $H(p) \le f(p) < H(p) + 1$

Greedy algorithms

$$\sum_{i=1}^n \rho_i |w_i|$$

- We denote the minimal value of $\sum_{i=1}^{n} p_i |w_i|$ by f(p). We define the entropy function to be: $H(p_1,...,p_n) = \sum_{j=1}^n p_j \log_2 \frac{1}{p_j}$ theorem: We have:
 - Shannon's theorem: We have:

Greedy algorithms

$$\sum_{i=1}^n p_i |w_i|$$

- We denote the minimal value of $\sum_{i=1}^{n} p_i |w_i|$ by f(p). We define the entropy function to be: $H(p_1, ..., p_n) = \sum_{i=1}^{n} p_i \log_2 \frac{1}{p_i}$
- Shannon's theorem: We have:

Greedy algorithms

• We denote the minimal value of $\sum_{i=1}^{n} p_i |w_i|$ by f(p). • We define the entropy function to be:

$$\sum_{i=1}^n p_i |w_i|$$

- $H(p_1, ..., p_n) = \sum_{i=1}^n p_i \log_2 \frac{1}{p_i}$
- <u>Shannon's theorem:</u> We have:

$$H(p) \le f(p) < H(p) + 1$$

Shannon's theorem - example

Greedy algorithms

• For $p=(\frac{1}{2},\frac{1}{4},\frac{1}{8},\frac{1}{8})$ we found a prefix code with $\sum_{i=1}^m p_i |w_i|=1.75$. It follows that $f(p)\leq 1.75$. On the other hand:

$$H(\rho) = \frac{1}{2}\log_2 2 + \frac{1}{4}\log_2 4 + \frac{1}{8}\log_2 8 + \frac{1}{8}\log_2 8 =$$

$$= 0.5 + 0.5 + \frac{3}{8} + \frac{3}{8} = 1.75$$

• It follows that the code we found is optimal $(1.75 = H(p) \le f(p)).$

Shannon's theorem - example

Greedy algorithms

• For $p=(\frac{1}{2},\frac{1}{4},\frac{1}{8},\frac{1}{8})$ we found a prefix code with $\sum_{i=1}^n p_i |w_i|=1.75$. It follows that $f(p)\leq 1.75$. On the other hand:

 $H(p) = \frac{1}{2}\log_2 2 + \frac{1}{4}\log_2 4 + \frac{1}{8}\log_2 8 + \frac{1}{8}\log_2 8 =$

$$= 0.5 + 0.5 + \frac{3}{8} + \frac{3}{8} = 1.75$$

• It follows that the code we found is optimal $(1.75 = H(p) \le f(p)).$

Shannon's theorem - proof (we will skip the proof)

Greedy algorithms

• Proof: Given $p = (p_1, ..., p_n)$ we define $l_i = \lceil \log_2 \frac{1}{p_i} \rceil$. We get that:

$$\sum_{i} 2^{-l_i} \le \sum_{i} 2^{-\log_2 \frac{1}{p_i}} = \sum_{i} p_i = 1$$

It follows that there exists a binary tree T with leaves $v_1,...,v_n$ such that $h(v_i)=l_i.$ Therefore:

$$\underbrace{f(\rho)}_{\text{optimal}} \leq \sum_{i} p_i \cdot I_i = \sum_{i=1}^n p_i \cdot \left\lceil \log_2 \frac{1}{p_i} \right\rceil \leq \\ \sum_{i=1}^n p_i \cdot \left(\log_2 \frac{1}{p_i} + 1 \right) = H(\rho) + 1$$

Shannon's theorem - proof (we will skip the proof)

• Let $w_1,...,w_n$ be a prefix code with lengths $l_1,...,l_n$. We know that $\sum_{i=1}^n 2^{-l_i} \le 1$. Therefore:

Greedy algorithms

$$0 \ge \log_2\left(\sum_{i=1}^n 2^{-l_i}\right) = \log_2\left(\sum_{i=1}^n p_i \frac{2^{-l_i}}{p_i}\right)$$

Existence of prefix codes Shannon's theorem Huffman algorithm

$$0 \ge \log_2\left(\sum_{i=1}^n 2^{-l_i}\right) = \log_2\left(\sum_{i=1}^n p_i \frac{2^{-l_i}}{p_i}\right)$$

$$\ge \sum_{i=1}^n p_i \log_2\frac{1}{p_i} + \sum_{i=1}^n p_i \log_2 2^{-l_i} = H(p) - \sum_{i=1}^n p_i l_i$$

(Note: the convexity here is actually concavity)

$$= \sum_{i=1}^{n} p_i \log_2 \frac{1}{p_i} + \sum_{i=1}^{n} p_i \log_2 2^{-l_i} = H(p) - \sum_{i=1}^{n} p_i$$

Greedy algorithms • So far we haven't said anything about how to find an optimal prefix code. Sharron's sharron's hufman algorithm: • The Huffman Algorithm: (for an optimal prefix code for $p = (p_1, ..., p_n)$) **Spanning trees Graphs (among trees for algorithm (amon

Greedy algorithms codes Greedy algorithms • So far we haven't said anything about how to find an optimal prefix code. Estimate the code optimal prefix code optimal prefix code optimal prefix code for $p = (p_1, ..., p_n)$ Spanning trees consists and consists and optimal prefix code for $p = (p_1, ..., p_n)$ Spanning trees consists and consists and

Huffman coding

Greedy algorithms

 So far we haven't said anything about how to find an optimal prefix code.

• The Huffman Algorithm:

 $\overline{(ext{for an optimal prefix code for } p = (p_1,...,p_n))}$

• Find the 2 minimal elements of $p=(p_1,...,p_n)$ (without loss of generality $p_1,...,p_{n-2}\geq p_{n-1},p_n$).
• Find (recursively) an optimal binary tree for

 $(p_1,...,p_{n-2},p_{n-1}+p_n).$ § Split the node (leaf) of $p_{n-1}+p_n$ to 2 nodes.

Greedy
algorithms
Huffman
codes
Existence of

 So far we haven't said anything about how to find an optimal prefix code.

The Huffman Algorithm:

 $\overline{(\text{for an optimal prefix code for }p=(p_1,...,p_n))}$

Find the 2 minimal elements of p = (p₁,..., p_n) (without loss of generality p₁,..., p_{n-2} ≥ p_{n-1}, p_n).
 Find (recursively) an optimal binary tree for

 $(p_1,...,p_{n-2},p_{n-1}+p_n).$

© Split the node (leaf) of $p_{n-1} + p_n$ to 2 nodes.

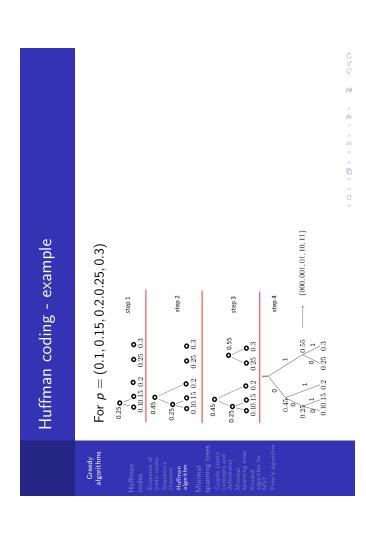
Greedy algorithms

• So far we haven't said anything about how to find an optimal prefix code.

The Huffman Algorithm:

 $\overline{(ext{for an optimal prefix code for } p = (p_1,...,p_n))}$

- U Find the 2 minimal elements of p = (p₁,..., p_n) (without loss of generality p₁,..., p_{n-2} ≥ p_{n-1}, p_n).
 Q Find (recursively) an optimal binary tree for (p₁,..., p_{n-2}, p_{n-1} + p_n).
 Split the node (leaf) of p_{n-1} + p_n to 2 nodes.



Greedy algorithms

codes Existence of prefix codes Shannon's

Minimal

Graphs (some concepts and definitions)
Minimal spanning trees
Kruskal

 \bullet The following theorem proves the correctness of the algorithm.

• Theorem: Let $p_1 \ge p_2 \ge ... \ge p_n$ be such that $\sum_{j=1}^n p_1 = 1$ ($p_i \ge 0$) then:

$$f(p_1, p_2, ..., p_n) = f(p_1, p_2, ..., p_{n-1} + p_n) + p_{n-1} + p$$

(Remember that f(p) is the value of $\sum_{i} p_{i} |w_{i}|$ whose, $(w_{1},...,w_{n})$ is an optimal prefix code).

Greedy algorithms

• The following theorem proves the correctness of the algorithm.

• Theorem: Let $p_1 \ge p_2 \ge ... \ge p_n$ be such that $\sum_{i=1}^n p_1 = 1$ ($p_i \ge 0$) then:

$$f(p_1, p_2, ..., p_n) = f(p_1, p_2, ..., p_{n-1} + p_n) + p_{n-1} + p_n$$

(Remember that f(p) is the value of $\sum_i p_i |w_i|$ where (w_1, \dots, w_n) is an optimal prefix code).

Greedy algorithms

proof: Let T' be an optimal tree for the probability vector $\overline{p'}=\overline{(p_1,...,p_{n-1}+p_n)}.$ Let $v_1,...,v_{n-2},v'_{n-1}$ be the leaves of T'. We build a new tree T by splitting the leaf v'_{n-1} into 2 nodes v_{n-1} and v_n .

$$\bigcup_{0}^{n-1+p_n}$$

 $\bigcap_{p_{n-1}+p_n} \bigcirc \bigcirc$ $\bigcap_{p_{n-1}-p_n} \bigcirc$ The average length for the tree T that we get is:

$$\left(\sum_{j=1}^{n-2} p_i h_{T'}(v_i)\right) + p_{n-1}(h_{T'}(v'_{n-1}) + 1) + p_n(h_{T'}(v'_{n-1}) + 1) =$$

$$\left(\sum_{j=1}^{n-1} p_i h_{T'}(v_i)\right) + p_{n-1} + p_n$$

Greedy algorithms proof: It follows that:

 $f(p_1, p_2, ..., p_n) \le f(p_1, ..., p_{n-2}, p_{n-1} + p_n) + p_{n-1} + p_n$

proof: On the other hand, let T be an optimal tree for $\overline{p}=(\overline{p}_1,...,p_n)$. If the leaves of T are $v_1,...,v_n$ then we have:

Greedy algorithms

$$f(p) = \sum_{i=1}^{i=n} p_i h_T(v_i)$$

Let i be such that the height of v_i is maximal and let T' be the tree that we get by exchanging v_i and v_n in T' (if i=n we do nothing).



Greedy algorithms

 $\overline{\text{proof:}}\ T'$ must be an optimal tree for $(p_1,...,p_n)$ since:

$$\sum_{i=1}^{i=n} p_i h_{\mathcal{T}}(v_i) - \sum_{i=1}^{i=n} p_i h_{\mathcal{T}'}(v_i) =$$

$$: h_{\mathcal{T}}(v_i) + \rho_{-}h_{\mathcal{T}}(v_{-}) - (\rho_i h_{\mathcal{T}}(v_{-}) + \rho_{-}h_{\mathcal{T}}(v_{-}))$$

$$(p_ih_{\mathcal{T}}(v_i) + p_nh_{\mathcal{T}}(v_n)) - (p_ih_{\mathcal{T}}(v_n) + p_nh_{\mathcal{T}}(v_i)) = \underbrace{(p_i - p_n)}_{\geq 0} \underbrace{(h_{\mathcal{T}}(v_i) - h_{\mathcal{T}}(v_n))}_{\geq 0} \geq 0$$

It follows that there exists an optimal tree T such that $h_T(\nu_n)$ is maximal (among all the leaves).

Clearly v_n is not the only child of its parent in T, since then we could remove one edge and improve the optimality of T.



Greedy algorithms

proof: It follows that there are 2 children.

Since V_n has maximal height, it follows that the other child is also of maximal height.

Similar arguments show that we can make sure that the other child is v_{n-1} .

 v_{n-1} have the same parent (and are of maximal height). We consider a new tree T'' with leaves $u_1, \dots, u_{n-2}, u_{n-1}$ where So far we saw that there exists an optimal ${\mathcal T}$ such that ν_n and $u_i = v_i$ and $u_{n-1} = \mathsf{parent}(v_n)$.

Greedy algorithms

proof: If we consider the frequencies vector $\overline{q=(q_1,...,q_{n-1})=(p_1,...,p_{n-2},p_{n-1}+p_n)} \text{ then we see that:}$

 $f(q) \le \sum_{i=1}^{n-1} h_{T''}(u_i)q_i = \sum_{i=1}^{n-2} h_T(v_i)p_i + (p_{n-1} + p_n)(\underbrace{h_T(v_n)}_{=h_T(v_{n-1})} - 1) = \underbrace{h_T(v_n)}_{=h_T(v_{n-1})}$

 $= \sum_{j=1}^{n} h_{T}(v_{j})p_{j} - (p_{n-1} + p_{n}) = f(p) - (p_{n-1} + p_{n})$

We see that:

 $f((p_1,...,p_{n-2},p_{n-1}+p_n))=f(q)\leq f(p)-(p_{n-1}+p_n)$



Greedy algorithms

• A graph is a set of vertices and edges that connect them.

Existence of prefix codes Shannon's theorem Huffman

Minimal
spanning trees
Graphs (some
concepts and
definitions)
Minimal
spanning trees

• More accurately, we have a set V of vertices (or nodes)

- and a set E of edges.
- An edge $e \in E$ is a pair of vertices $e = \{v,u\}$, $u,v \in V$. We denote G = (V,E).
- A graph might be directed. In a directed graph the edges have a direction. The edges (u, v) and (v, u) are two different edges.

Graphs Greedy algorithms • A graph is a set of vertices and edges that connect them. Hufman codes Existence of prefix codes Shamon's therem with the same of prefix codes Shamon's there are consistence of the connect them. • More accurately, we have a set V of vertices (or nodes) and a set E of edges. • An edge $e \in E$ is a pair of vertices $e = \{v, u\}$, $u, v \in V$. • A graph might be directed. In a directed graph the edges

Graphs Greedy algorithms A graph is a solution odes codes codes instruction instruction

A graph is a set of vertices and edges that connect them.

 More accurately, we have a set V of vertices (or nodes) and a set E of edges. • An edge $e \in E$ is a pair of vertices $e = \{v, u\}$, $u, v \in V$.

• We denote G = (V, E).

• A graph might be directed. In a directed graph the edges have a direction. The edges (u,v) and (v,u) are two different edges.

Greedy algorithms A graph is a set of vertices and edges that connect them.



- More accurately, we have a set V of vertices (or nodes) and a set E of edges.
- $\bullet \ \, {\rm An\ edge}\,\, e \in E \,\, {\rm is}\,\, {\rm a\ pair\ of\ vertices}\,\, e = \{\nu,u\}\,\,,\,\, u,v \in V.$
 - We denote G = (V, E).
- A graph might be directed. In a directed graph the edges have a direction. The edges (u,v) and (v,u) are two different edges.

Greedy algorithms codes
Existence of
prefix codes
Shamon's
theorem
Huffman

Minimal spanning trees Graphs (some concepts and definitions) Minimal spanning trees

More accurately, we have a set V of vertices (or nodes)

• A graph is a set of vertices and edges that connect them.

- and a set E of edges. • An edge $e \in E$ is a pair of vertices $e = \{v, u\}$, $u, v \in V$.
 - We denote G = (V, E).
- A graph might be directed. In a directed graph the edges have a direction. The edges (u, v) and (v, u) are two different edges.



Greedy algorithms

We say that a graph G=(V,E) is connected, if for any 2 vertices $u,v\in V$ there is a path from u to v: $u=v_0,v_1,...,v_n=v \text{ such that }$ $(v_i,v_{i+1})\in E \text{ for } 0\leq i\leq n-1$

• A simple cycle in a graph is a closed walk from one vertex to itself with no repetitions of vertices and edges except the first and last vertices:

$$l=v_0,v_1,...,v_n=u$$
 s.t $(v_i,v_{i+1})\in E$ and $v_i
eq$

for any
$$i \neq j$$
 except $i = 0, j =$

Greedy algorithms

• We say that a graph G=(V,E) is connected, if for any 2 vertices $u,v\in V$ there is a path from u to v:

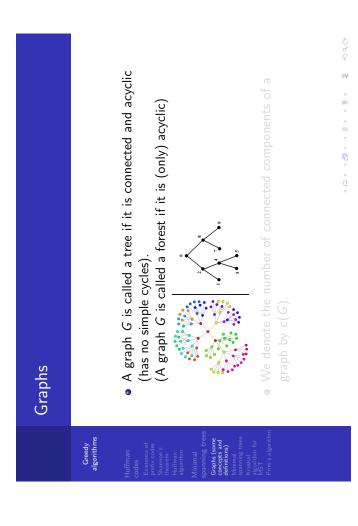
$$u=\nu_0,\, \nu_1,...,\, \nu_n=\nu$$
 such that

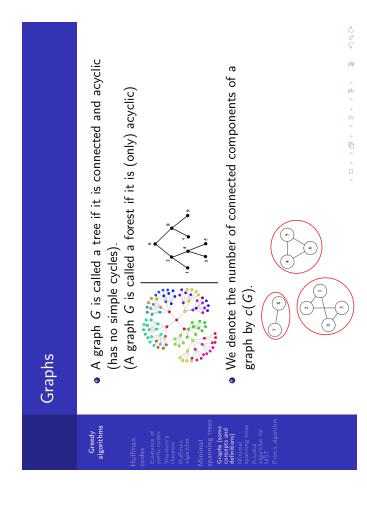
$$(v_i, v_{i+1}) \in E$$
 for $0 \le i \le n-1$

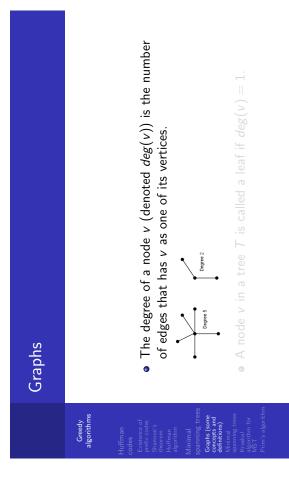
$$u=v_0,v_1,...,v_n=u$$
 s.t $\left(v_i,v_{i+1}
ight)\in E$ and $v_i \neq v_j$

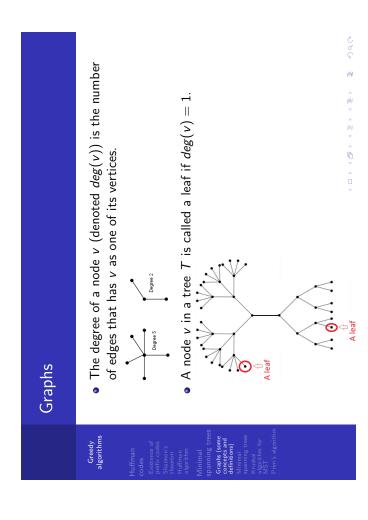
for any
$$i \neq j$$
 except $i = 0, j = n$

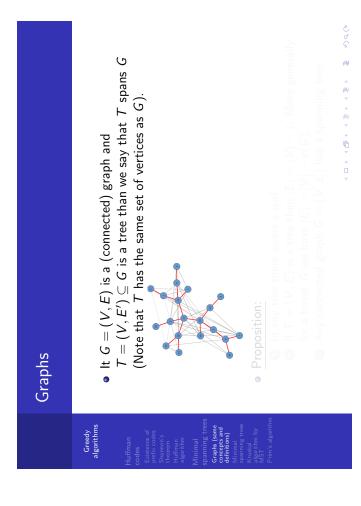












Greedy algorithms

• It G = (V, E) is a (connected) graph and $T = (V, E') \subseteq G$ is a tree than we say that T spans G (Note that T has the same set of vertices as G).



- Proposition:
- In any tree there exists a leaf. If T=(V,E) is a tree then |E|=|V|-1. More generally, for a forest G we have |E|=|V|-c(G). Any connected graph G=(V,E) has a spanning tree.

Graphs Greedy algorithms

• It G = (V, E) is a (connected) graph and $T = (V, E') \subseteq G$ is a tree than we say that T spans G (Note that T has the same set of vertices as G).

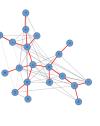


- Proposition:
- In any tree there exists a leaf.
- ① If T=(V,E) is a tree then |E|=|V|-1. More generally, for a forest G we have |E|=|V|-c(G). ③ Any connected graph G=(V,E) has a spanning tree.



Greedy algorithms

• It G = (V, E) is a (connected) graph and $T = (V, E') \subseteq G$ is a tree than we say that T spans G (Note that T has the same set of vertices as G).



- Proposition:
- ① In any tree there exists a leaf. ② If T=(V,E) is a tree then |E|=|V|-1. More generally, for a forest G we have |E|=|V|-c(G).
 - \odot Any connected graph G=

Greedy algorithms

• It G = (V, E) is a (connected) graph and $T = (V, E') \subseteq G$ is a tree than we say that T spans G (Note that T has the same set of vertices as G).

- Proposition:
- In any tree there exists a leaf.
 If T = (V, E) is a tree then |E| = |V| 1. More generally, for a forest G we have |E| = |V| c(G).
 Any connected graph G = (V, E) has a spanning tree.

Greedy algorithms

codes Existence of prefix codes Shannon's theorem

Minimal spanning tr Graphs (som definitions)
Minimal spanning trees
Kruskal algorithm for MST

• Proof:

- Choose some node v_0 . If v_0 is a leaf we're done. Otherwise we can move from v_0 to some v_1 . If v_1 is a leaf we're done. Continuing like this, we never return to a node that we already visited since a tree has no cycles. It follows that (at some point) we must reach a leaf.
- We prove by induction on |V|. If |V|=1 than clearly, |E|=0 and c(G)=1. Let $T\subseteq G$ be a connected component (T must be a tree). It follows from (1) that T has a leaf. Let v_0 be a leaf of T. If we remove v_0 from G we get a graph G'=(V',E') with |V'|=|V|-1, |E'|=|E|-1 and c(G')=c(G) (If T is just one vertex then |V'|=|V|-1, |E'|=|E| and c(G')=c(G)-1). Therefore:

$$c(G) = c(G') = |V'| - |E'| = |V| - |E|$$

Proof:

Greedy algorithms

Choose some node v_0 . If v_0 is a leaf we're done. Otherwise we can move from v_0 to some v_1 . If v_1 is a leaf we're done. Continuing like this, we never return to a node that we already visited since a tree has no cycles. It follows that (at some point) we must reach a leaf.

We prove by induction on |V|. If |V|=1 than clearly, |E|=0 and c(G)=1. Let $T\subseteq G$ be a connected component (T must be a tree) It follows from (1) that T has a leaf. Let v_0 be a leaf of T. If we remove v_0 from G we get a graph G'=(V',E') with |V'|=|V|-1, |E'|=|E|-1 and c(G')=c(G) (If T is just one vertex then |V'|=|V|-1, |E'|=|E| and c(G')=c(G)-1). Therefore:

$$c(G) = c(G') = |V'| - |E'| = |V| - |E|$$

Greedy algorithms

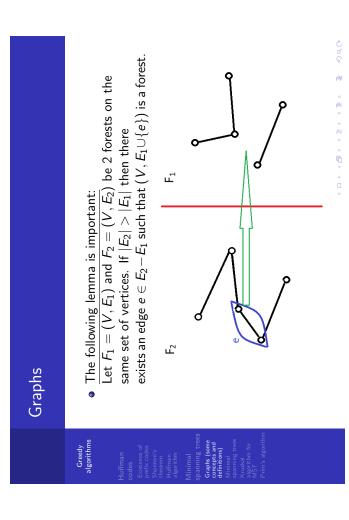
Proof:

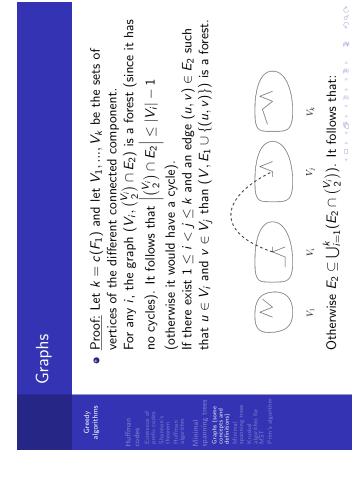
- Choose some node v_0 . If v_0 is a leaf we're done. Otherwise we can move from v_0 to some v_1 . If v_1 is a leaf we're done. Continuing like this, we never return to a node that we already visited since a tree has no cycles. It follows that (at some point) we must reach a leaf.
 - We prove by induction on |V|. If |V|=1 than clearly, |E|=0 and c(G)=1. Let $T\subseteq G$ be a connected component (T must be a tree). It follows from (1) that T has a leaf. Let v_0 be a leaf of T. If we remove v_0 from G we get a graph G'=(V',E') with |V'|=|V|-1, |E'|=|E|-1 and c(G')=c(G) (If T is just one vertex then |V'|=|V|-1, |E'|=|E| and c(G')=c(G)-1). Therefore:

$$c(G) = c(G') = |V'| - |E'| = |V| - |E|$$

Greedy algorithms Huffman codes
Enstence of preference of preference of preference of the codes
Shamon's Huffman algorithm
Multimal spanning trees
Gaphs (some concepts and definitions)
Minimal spanning trees
Krustal algorithm for Multimal

• Proof: (3) By induction. Choose $v_0 \in V$ and remove it from G. By the induction hypothesis each connected component in $G - \{v_0\}$ has a minimal spanning tree. Let $T_1, ..., T_m$ be the spanning trees for the connected components of $G' = G - \{v_0\}$. Since G is connected, there exists an edge from (at least) one of the nodes in each of the trees $T_1, ..., T_m$ to v_0 . Adding one such edge from each T_i to v_0 we get a spanning tree T for G.





Greedy algorithms ſ

uffman gorithm inimal anning trees

spanning trees
Graphs (some
concepts and
definitions)
Minimal
spanning trees
Kruskal

• Proof:

 $|E_2| \le \sum_{i=1}^k \left| E_2 \cap \binom{V_i}{2} \right| \le \sum_{i=1}^k \left(|V_i| - 1 \right) = |V| - k = |E_1|$

But this is a contradiction.

Minimal spanning trees

Greedy algorithms

• The problem: We are given a connected graph G=(V,E) with different weights on its edges. The weights are given by $w: E \to \mathbb{R}_{>0}$. We wish to find a spanning tree T = (V, E') of minimal weight. A spanning tree T with minimal $\sum_{e \in E'} w(e)$ is called a Minimal Spanning Tree (MST).

• Note that a minimal spanning tree is not necessarily

Minimal spanning trees

Greedy algorithms

Huffman codes Existence of prefix codes Shannon's theorem Huffman algorithm

Minimal
Spanning trees
Graphs (some
Graphs (some
definitions)
Minimal
spanning trees
Aftenskal
algorithm for
MK-TH

• The problem: We are given a connected graph G = (V, E) with different weights on its edges. The weights are given by $w: E \to \mathbb{R}_{>0}$. We wish to find a spanning tree T = (V, E') of minimal weight. A spanning tree T with minimal $\sum_{e \in E'} w(e)$ is called a Minimal Spanning Tree (MST).

 Note that a minimal spanning tree is not necessarily unique.

Greedy algorithms

Kruskal algorithm for MST

• Initialize: find $e_1 \in E$ with minimal weight. $w(e_1) = \min_{e \in E} \{w(e)\}.$

• Step: After choosing $e_1, ..., e_k$, we choose e_{k+1} to be an

• Finish: Once we choose e_{n-1} we are done.

• We will prove: Let T'=(V,F) be some spanning tree of \overline{G} (not necessarily a minimal one). If we order the edges

• It follows immediately from the above proposition that the

Greedy algorithms

- Kruskal algorithm finds a minimal spanning tree for a connected graph G = (V, E) (we denote |V| = n):
 Initialize: find e₁ ∈ E with minimal weight.
 - $w(e_1) = \min_{e \in E} \{w(e)\}.$
- Step: After choosing $e_1, ..., e_k$, we choose e_{k+1} to be an edge of minimal weight such that $(V, \{e_1, ..., e_k, e_{k+1}\})$ is acyclic.
 - Finish: Once we choose e_{n-1} we are done.
- We will prove: Let T' = (V, F) be some spanning tree of \overline{G} (not necessarily a minimal one). If we order the edges of T' as $(f_1, ..., f_{n-1})$ such that

Kruskal algorithm for MST

$$w(e_k) \le w(f_k)$$
 for any $1 \le k \le n$

• It follows immediately from the above proposition that the algorithm actually finds a MST.

Greedy algorithms

- Kruskal algorithm finds a minimal spanning tree for a connected graph G = (V, E) (we denote |V| = n):
 Initialize: find e₁ ∈ E with minimal weight.
 - $w(e_1) = \min_{e \in E} \{w(e)\}.$
- Step: After choosing e₁, ..., e_k, we choose e_{k+1} to be an edge of minimal weight such that (V, {e₁, ..., e_k, e_{k+1}}) is acyclic.
 - Finish: Once we choose e_{n-1} we are done.
- We will prove: Let T' = (V, F) be some spanning tree of G (not necessarily a minimal one). If we order the edges of T' as $(f_1, ..., f_{n-1})$ such that

Kruskal algorithm for MST

$$w(e_k) \le w(f_k)$$
 for any $1 \le k \le n -$

• It follows immediately from the above proposition that the above proposition that the

Greedy algorithms

- Kruskal algorithm finds a minimal spanning tree for a connected graph G = (V, E) (we denote |V| = n):
 Initialize: find e₁ ∈ E with minimal weight.
 - $w(e_1) = \min_{e \in E} \{w(e)\}.$
- $\overline{\text{edge}}$ of minimal weight such that ($V,\{e_1,...,e_k,e_{k+1}\})$ is • Step. After choosing $e_1, ..., e_k$, we choose e_{k+1} to be an acyclic.
 - Finish: Once we choose e_{n-1} we are done.
- We will prove: Let T'=(V,F) be some spanning tree of \overline{G} (not necessarily a minimal one). If we order the edges

Kruskal algorithm for MST

$$w(e_k) \le w(f_k)$$
 for any $1 \le k \le n$

• It follows immediately from the above proposition that the

Greedy algorithms

- Kruskal algorithm finds a minimal spanning tree for a connected graph G = (V, E) (we denote |V| = n):
 Initialize: find e₁ ∈ E with minimal weight.
 w(e₁) = min_{e∈E} {w(e)}.
- $\overline{\text{edge}}$ of minimal weight such that ($V,\{e_1,...,e_k,e_{k+1}\})$ is • Step. After choosing $e_1, ..., e_k$, we choose e_{k+1} to be an acyclic.
 - Finish: Once we choose e_{n-1} we are done.
- We will prove: Let T' = (V, F) be some spanning tree of ${\cal G}$ (not necessarily a minimal one). If we order the edges of T' as $(f_1,...,f_{n-1})$ such that $w(f_1) \le w(f_2) \le ... \le w(f_{n-1})$ then we have:

 $w(e_k) \le w(f_k)$ for any $1 \le k \le n-1$

• It follows immediately from the above proposition that the

Greedy algorithms

Kruskal algorithm finds a minimal spanning tree for a connected graph G = (V, E) (we denote |V| = n):
 Initialize: find e₁ ∈ E with minimal weight.
 w(e₁) = min_{e∈E} {w(e)}.

edge of minimal weight such that $(V, \{e_1, ..., e_k, e_{k+1}\})$ is • Step. After choosing $e_1, ..., e_k$, we choose e_{k+1} to be an acyclic.

• $\overline{\text{Finish:}}$ Once we choose e_{n-1} we are done.

• We will prove: Let T' = (V, F) be some spanning tree of ${\cal G}$ (not necessarily a minimal one). If we order the edges of T' as $(f_1,...,f_{n-1})$ such that $w(f_1) \le w(f_2) \le ... \le w(f_{n-1})$ then we have:

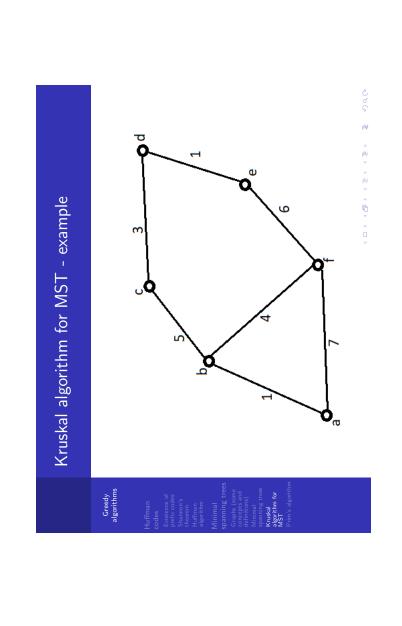
 $w(e_k) \le w(f_k)$ for any $1 \le k \le n-1$

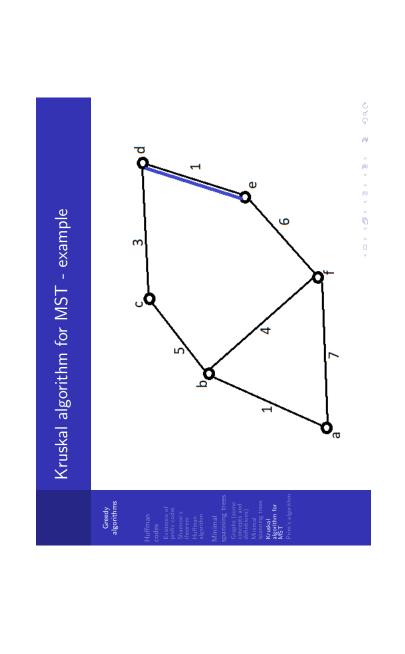
• It follows immediately from the above proposition that the algorithm actually finds a MST.

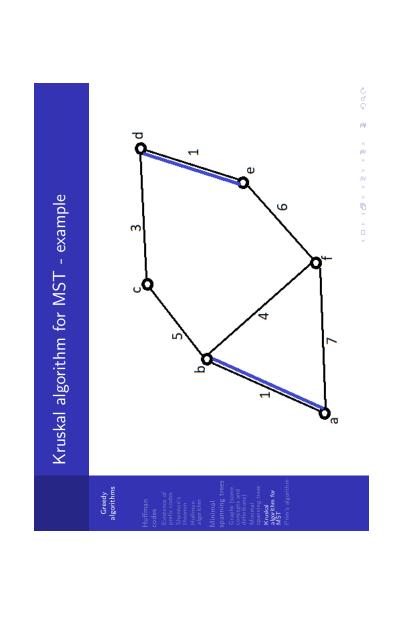
Greedy algorithms codes Existence of prefix codes Shamon's theorem Huffman algorithm

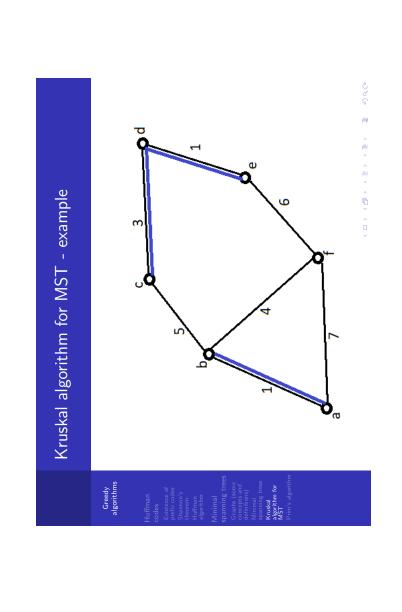
• Proof: For f_1 and e_1 it follows from the definition of e_1 . Denote $F_k = (f_1,...,f_k)$ and $E_{k-1} = (e_1,...,e_{k-1})$. The graphs (V,F_k) and (V,E_{k-1}) are both forests. Since $|F_k| > |E_{k-1}|$ it follows from the lemma we proved that there exists $f_i \in F_k$ such that $E_{k-1} \cup \{f_i\}$ is a forest. Therefore, by the way we choose e_k it follows that:

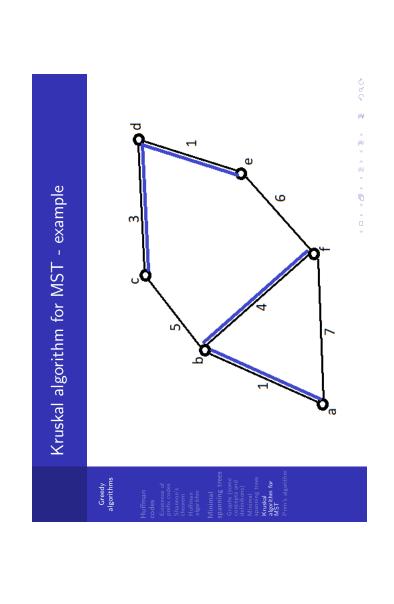
$$w(e_k) \leq w(f_i) \leq w(f_k)$$

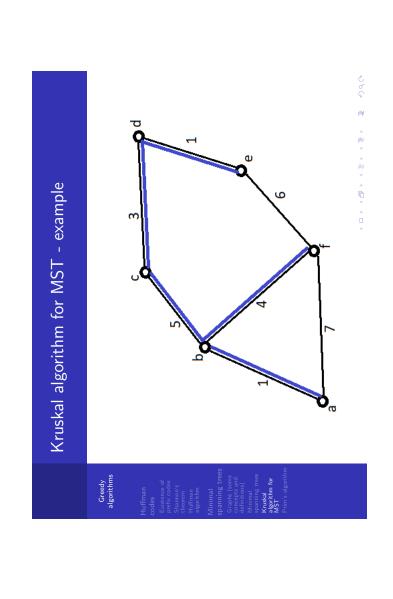


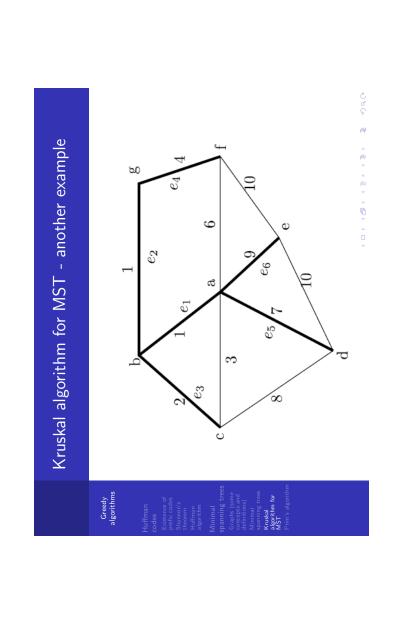












Greedy algorithms

How do we implement the algorithm?

- Through the algorithm, we keep track of the connected
- We start by initializing $E' = \emptyset$ and $A(\nu) = \{\nu\}$ for any
- We sort the edges by their weights:
- For $1 \le k \le m$: Let $e_k = (u, v)$. If $A(u) \ne A(v)$ then $E' = E' \cup \{e_k\}$ and:

Greedy algorithms Huffman

- How do we implement the algorithm?
- Through the algorithm, we keep track of the connected components. At each step, we will denote the connected component of a vertex ν by $A(\nu)$.
 - We start by initializing $E' = \emptyset$ and $A(v) = \{v\}$ for any $v \in V$.
 - We sort the edges by their weights:
 - $w(e_1) \leq w(e_2) \leq ... \leq w(e_m).$
- For $1 \le k \le m$: Let $e_k = (u, v)$. If $A(u) \ne A(v)$ then $E' = E' \cup \{e_k\}$ and:

 $A(x) = A(v) \cup A(v)$ for any $x \in A(u) \cup A(v)$

Greedy algorithms

 \bullet Through the algorithm, we keep track of the connected

components. At each step, we will denote the connected

component of a vertex ν by $A(\nu)$.

How do we implement the algorithm?

• We start by initializing $E' = \emptyset$ and $A(\nu) = \{\nu\}$ for any

• We sort the edges by their weights: $w(e_1) \leq w(e_2) \leq ... \leq w(e_m).$

- For $1 \le k \le m$: Let $e_k = (u, v)$. If $A(u) \ne A(v)$ then $E' = E' \cup \{e_k\}$ and:

Greedy algorithms

- How do we implement the algorithm?
- Through the algorithm, we keep track of the connected components. At each step, we will denote the connected component of a vertex ν by $A(\nu)$.
- We start by initializing $E' = \emptyset$ and $A(v) = \{v\}$ for any $v \in V$.
- We sort the edges by their weights: $w(e_1) \le w(e_2) \le ... \le w(e_m).$
- For $1 \le k \le m$: Let $e_k = (u, v)$. If $A(u) \ne A(v)$ then $E' = E' \cup \{e_k\}$ and:

 $A(x) = A(v) \cup A(v)$ for any $x \in A(u) \cup A(v)$

Greedy algorithms

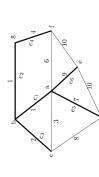
- How do we implement the algorithm?
- components. At each step, we will denote the connected \bullet Through the algorithm, we keep track of the connected component of a vertex ν by $A(\nu)$.
- We start by initializing $E' = \emptyset$ and $A(\nu) = \{\nu\}$ for any
- We sort the edges by their weights: $w(e_1) \le w(e_2) \le ... \le w(e_m).$
- For $1 \le k \le m$: Let $e_k = (u, v)$. If $A(u) \ne A(v)$ then $E' = E' \cup \{e_k\}$ and:

$$A(x) = A(\nu) \cup A(\nu)$$
 for any $x \in A(u) \cup A(\nu)$

Kruskal algorithm for MST - example

For the example we saw we get:

Greedy algorithms



Connected components Edges Remark	b	ab,c,d,e,f,g $ab=1$	abg.c,d,e,f $bg=1$	abgc,d,e,f bc = 2	ac = 3 In the same component	gf abgcf,d,e $gf = 4$	af = 6 In the same component	gf,ad abgcfd,e ad = 7	cd = 8 In the same component	f_1 , ad, ae f_2 Here we can finish	et $= 10$ $ $
ш	0	ab	ab, bg	ab, bg, bc		ab, bg, bc, gf		ab, bg, bc, gf, ad		ab, bg, bc, gf, ad, ae	

Greedy algorithms Huffman codes

- At each step of the algorithm we hold n sets of connected components $A_1, ..., A_n$ (might be empty).
- In the beginning $|A_i|=1$. In each step we have $t_i=|A_i|$ (this is just notatic For each $v\in V$ we maintain i(v) such that $v\in A$
- In the k-th step we check the k-th small e_k = (u, v);
- If l(u) == l(v) go to the next edge.

 If $l(u) \neq l(v)$ If $t_{l(u)} > t_{l(v)}$ we move $A_{l(v)}$ into A
- Counting operation:

Kruskal algorithm for MST Sorting $\rightarrow O(|E|\log|E|)$.

comparing i(u) and $i(v) \rightarrow |E|$

operations are performed on the next slide.

Greedy

algorithms

Component

Component

Component

Component

In the beging

In each ste

For each v

For each v

In the k-th

Initial

Component

Compo

• At each step of the algorithm we hold n sets of connected components $A_1, ..., A_n$ (might be empty).

• In the beginning $|A_i|=1$. In each step we have $t_i=|A_i|$ (this is jus

In each step we have $t_i=|A_i|$ (this is just notation). For each $v\in V$ we maintain i(v) such that $v\in A_{i(v)}$.

• In the *k*-th step we check the *k*-th smallest edge $e_k = (u, v)$:

• If $i(u) \neq i(v)$ - If $t_{i(u)} > t_{i(v)}$ we move $A_{i(v)}$ into $A_{i(u)}$ and we update all of the relevant A's and i(*)'s.

Counting operation:

Sorting $\rightarrow O(|E| \log |E|)$.

Comparing i(u) and i(v)
ightharpoonup |E|.

operations are performed on the next slide.

Greedy algorithms

ullet At each step of the algorithm we hold n sets of connected components $A_1, ..., A_n$ (might be empty).

• In the beginning $|A_i|=1$.

For each $v \in V$ we maintain i(v) such that $v \in A_{i(v)}$. In each step we have $t_i = |A_i|$ (this is just notation).

• In the k-th step we check the k-th smallest edge

 $e_k = (u, v)$:

① If i(u) == i(v) - go to the next edge. ② If $i(u) \neq i(v)$ - If $t_{i(u)} > t_{i(v)}$ we move $A_{i(v)}$ into $A_{i(u)}$ and we update all of the relevant A's and i(*)'s.

Counting operation:

Greedy algorithms

- At each step of the algorithm we hold n sets of connected components $A_1, ..., A_n$ (might be empty).
- In the beginning $|A_i|=1$. In each step we have $t_i=|A_i|$ (this is just notation). For each $v\in V$ we maintain i(v) such that $v\in A_{i(v)}$.
 - In the *k*-th step we check the *k*-th smallest edge $e_k = (u, v)$:
- ① If i(u) == i(v) go to the next edge. ② If $i(u) \neq i(v)$ - If $t_{i(u)} > t_{i(v)}$ we move $A_{i(v)}$ into $A_{i(u)}$ and we update all of the relevant A's and i(*)'s.
 - Counting operation: Sorting $\rightarrow O(|E|\log|E|)$. Comparing i(u) and $i(v) \rightarrow |E|$. Step 2 is done for |V|-1 edges. We will count how

Greedy algorithms

ullet At each step of the algorithm we hold n sets of connected components $A_1, ..., A_n$ (might be empty).

For each $v \in V$ we maintain i(v) such that $v \in A_{i(v)}$. In each step we have $t_i = |A_i|$ (this is just notation). ullet In the beginning $|A_i|=1.$

• In the k-th step we check the k-th smallest edge $e_k = (u, v)$:

① If i(u) == i(v) - go to the next edge. ② If $i(u) \neq i(v)$ - If $t_{i(u)} > t_{i(v)}$ we move $A_{i(v)}$ into $A_{i(u)}$ and we update all of the relevant A's and i(*)'s.

Counting operation:

Greedy algorithms

- ullet At each step of the algorithm we hold n sets of connected components $A_1, ..., A_n$ (might be empty).
- For each $v \in V$ we maintain i(v) such that $v \in A_{i(v)}$. In each step we have $t_i = |A_i|$ (this is just notation). ullet In the beginning $|A_i|=1.$
 - In the k-th step we check the k-th smallest edge
 - $e_k = (u, v)$:
- ① If i(u)==i(v) go to the next edge. ② If $i(u)\neq i(v)$ If $t_{i(u)}>t_{i(v)}$ we move $A_{i(v)}$ into $A_{i(u)}$ and we update all of the relevant A's and i(*)'s.
- Counting operation:

Sorting $\rightarrow O(|E| \log |E|)$.

Comparing i(u) and i(v) o |E|.

Step 2 is done for |V|-1 edges. We will count how many operations are performed on the next slide.

Greedy algorithms

Kruskal algorithm for MST

ullet Claim: Each vertex is being moved / updated $O(\log |V|)$

• <u>Proof:</u> Note that if we move $A_{i(u)}$ to $A_{i(v)}$ then we get a set at least twice the size of $A_{i(u)}$. It follows that a vertex can not move more than $\log_2 |V|$ times.

• It follows that overall the complexity is:

• Note that $|E| < |V|^2$ so $\log |E| < \log |V|^2 = 2\log |V|$. We

Greedy algorithms

- Claim: Each vertex is being moved / updated $O(\log |V|)$ times.
- <u>Proof:</u> Note that if we move $A_{i(u)}$ to $A_{i(v)}$ then we get a set at least twice the size of $A_{i(u)}$. It follows that a vertex can not move more than $\log_2 |V|$ times.
- Even if for all the vertices we perform $\log |V|$ operations, we are still bounded by $O(|V|\log |V|)$.
- It follows that overall the complexity is:

$$O(|E|\log|E|) + O(|E|) + O(|V|\log|V|) = O(|E|\log|E|)$$

Kruskal algorithm for MST • Note that $|E|<|V|^2$ so $\log |E|<\log |V|^2=2\log |V|$. We see that:

 $O(|E|\log|E|) = O(|E|\log|V|$

Greedy algorithms • [t]

 \bullet <u>Claim:</u> Each vertex is being moved / updated $O(\log |V|)$ times.

• Proof: Note that if we move $A_{i(u)}$ to $A_{i(v)}$ then we get a set at least twice the size of $A_{i(u)}$. It follows that a vertex can not move more than $\log_2 |V|$ times.

• Even if for all the vertices we perform $\log |V|$ operations, we are still bounded by $O(|V|\log |V|)$.

• It follows that overall the complexity is:

 $O(|E|\log|E|) + O(|E|) + O(|V|\log|V|) = O(|E|\log|E|)$

Kruskal algorithm for MST • Note that $|E| < |V|^2$ so $\log |E| < \log |V|^2 = 2\log |V|$. We see that:

 $O(|E| \log |E|) = O(|E| \log |V|)$

 \bullet Claim: Each vertex is being moved / updated $O(\log |V|)$ times.

Greedy algorithms • <u>Proof:</u> Note that if we move $A_{i(u)}$ to $A_{i(v)}$ then we get a set at least twice the size of $A_{i(u)}$. It follows that a vertex can not move more than $\log_2 |V|$ times.

 \bullet Even if for all the vertices we perform $\log |V|$ operations, we are still bounded by $O(|V|\log |V|).$

• It follows that overall the complexity is:

$$O(|E|\log|E|) + O(|E|) + O(|V|\log|V|) = O(|E|\log|E|)$$

• Note that $|E| < |V|^2$ so $\log |E| < \log |V|^2 = 2\log |V|$. We see that:

 $O(|E| \log |E|) = O(|E| \log |V|)$

Greedy algorithms

- \bullet <u>Claim:</u> Each vertex is being moved / updated $O(\log |V|)$ times.
- <u>Proof:</u> Note that if we move $A_{i(u)}$ to $A_{i(v)}$ then we get a set at least twice the size of $A_{i(u)}$. It follows that a vertex can not move more than $\log_2 |V|$ times.
 - \bullet Even if for all the vertices we perform $\log |V|$ operations, we are still bounded by $O(|V|\log |V|).$
- It follows that overall the complexity is:

$$O(|E|\log|E|) + O(|E|) + O(|V|\log|V|) = O(|E|\log|E|)$$

 • Note that $|E| < |V|^2$ so $\log |E| < \log |V|^2 = 2\log |V|$. We see that:

$$O(|E|\log |E|) = O(|E|\log |V|)$$



Prim's algorithm

Greedy algorithms

• We will now see Prim's algorithm for finding a MST for a graph G = (V, E).

- The idea is:
- Choose a random node v_1 and define $V_1 = \{v_1\}$ and $E_1 = \emptyset$. Define inductively a sequence of **trees** $T_k = (V_k, E_k)$ where $V_k = \{v_1, ..., v_k\} \subseteq V$ and $E_k = \{e_1, ..., e_{k-1}\} \subseteq E$

Prim's algorithm

Greedy algorithms

 $oldsymbol{\mathbb{Q}}$ Choose a random node v_1 and define $V_1=\{v_1\}$ and $E_1=\emptyset$.

• We will now see Prim's algorithm for finding a MST for a

graph G = (V, E).

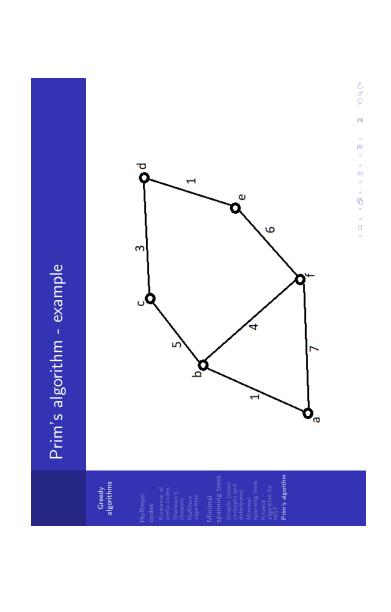
The idea is:

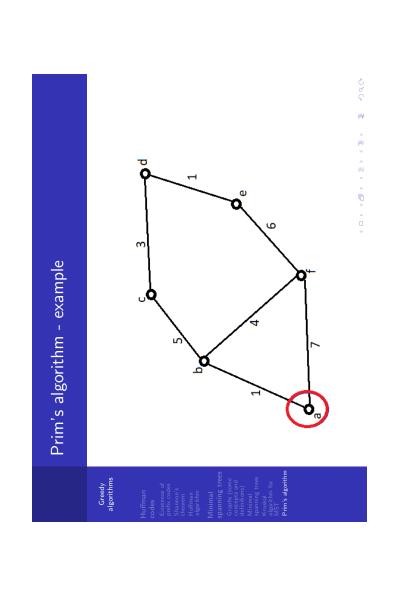
© Define inductively a sequence of **trees** $T_k = (V_k, E_k)$ where $V_k = \{v_1, ..., v_k\} \subseteq V$ and $E_k = \{e_1, ..., e_{k-1}\} \subseteq E$

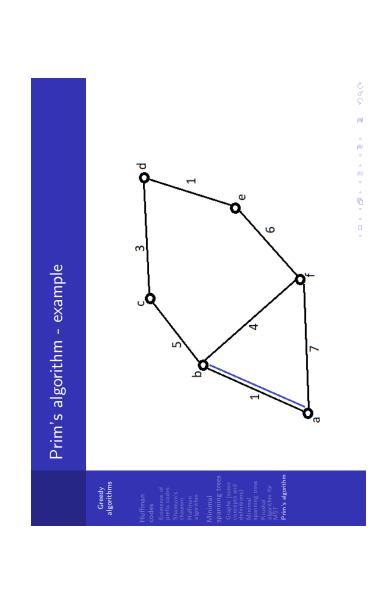
Prim's algorithm

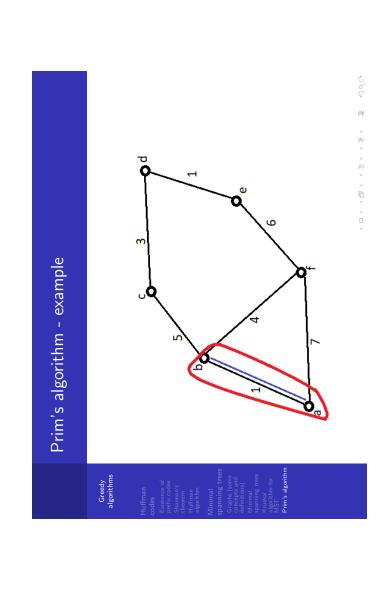
Greedy algorithms

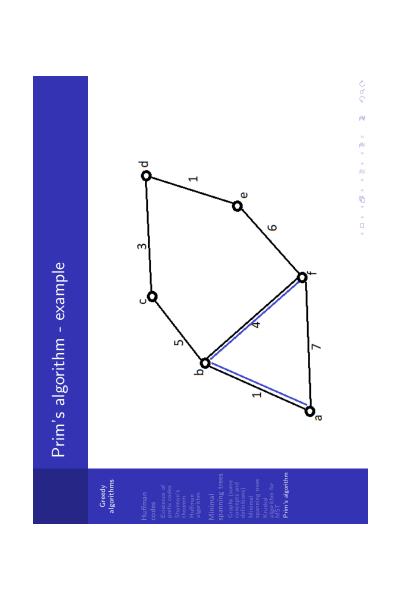
- We will now see Prim's algorithm for finding a MST for a graph G = (V, E).
- The idea is:
- lacktriangle Choose a random node v_1 and define $V_1=\{v_1\}$ and
- $E_1=\emptyset.$ Define inductively a sequence of **trees** $T_k=(V_k,E_k)$ where $V_k=\{v_1,...,v_k\}\subseteq V$ and $E_k=\{e_1,...,e_{k-1}\}\subseteq E$
- in the following way: Let $E(V_k, V V_k)$ be all the edges $\{u, v\}$ of G such that $u \in V_k$ and $v \in V V_k$ and let $e = \{u, v\} \in E(V_k, V V_k)$ be an edge with minimal weight in $E(V_k, V V_k)$.
- Define $e_k=e$, $v_{k+1}=v$ and $V_{k+1}=V_k\cup\{v_{k+1}\}$, $E_{k+1} = E_k \cup \{e_k\}.$ Set $T_{k+1} = (V_{k+1}, E_{k+1}).$

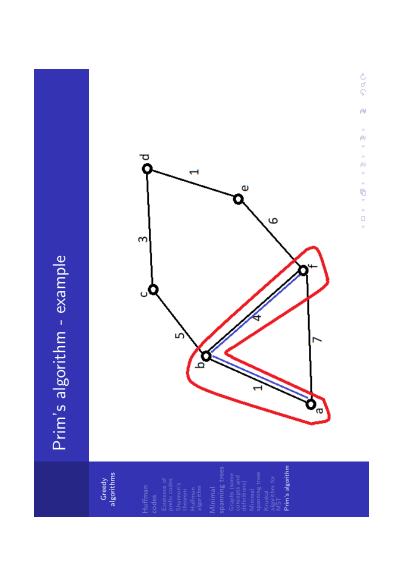


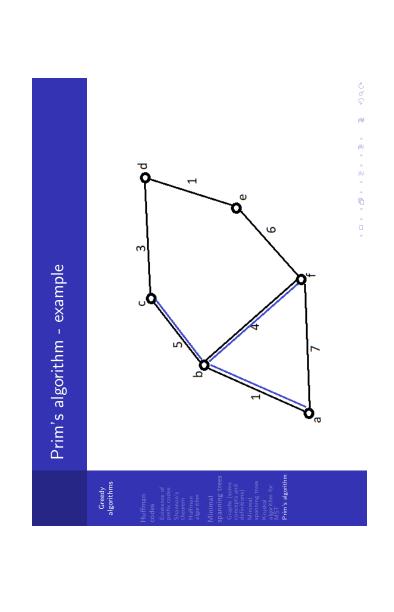


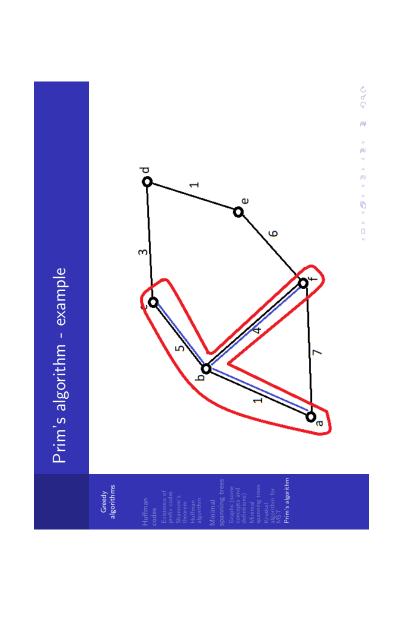


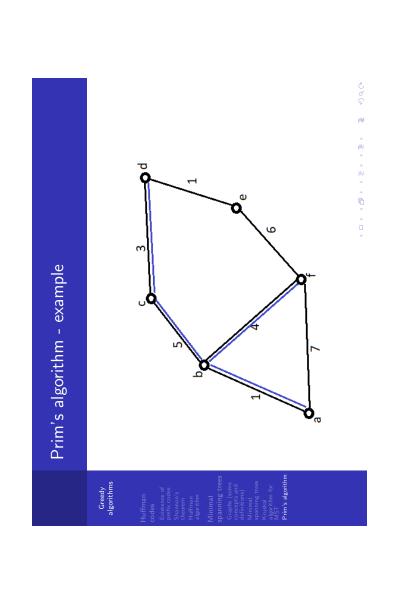


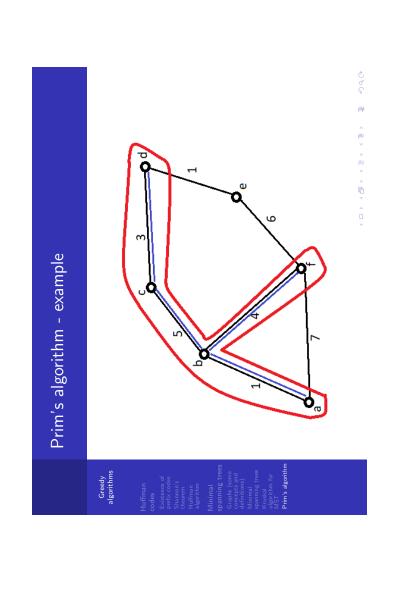


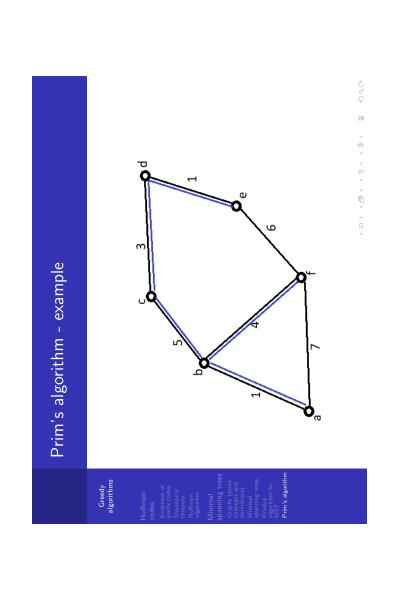












Greedy algorithms

codes Existence of prefix codes

Huffman algorithm

Graphs (some concepts and definitions) Minimal spanning trees Kruskal algorithm for

• Claim: For any $k \in \{1,2,...,n=|V|\}$, the tree T_k is contained in a Minimal Spanning Tree of G.

- \bullet The correctness of Prim's algorithm follows immediately since for k=n we get that T_n is a MST.
- Proof: We prove by induction on k.
 For k = 1, there is only one vertex which is clearly in some MST.

By our induction hypothesis there exists a MST T = (V, F) such that $T_k \subseteq T$. If $e_k \in F$ than T_{k+1} g and we're done.

Greedy algorithms

Huffman codes Existence of prefix codes

Minimal spanning tre

Graphs (some concepts and definitions)
Minimal spanning trees Kruskal algorithm for MST

• Claim: For any $k \in \{1,2,...,n=|V|\}$, the tree T_k is contained in a Minimal Spanning Tree of G.

- The correctness of Prim's algorithm follows immediately since for k=n we get that T_n is a MST.
- Proof: We prove by induction on k. For k=1, there is only one vertex which is clearly in some MST

By our induction hypothesis there exists a MST T = (V, F) such that $T_k \subseteq T$. If $e_k \in F$ than $T_{k+1} \subseteq F$

Greedy algorithms

• Claim: For any $k \in \{1, 2, ..., n = |V|\}$, the tree T_k is contained in a Minimal Spanning Tree of G. • The correctness of Prim's algorithm follows immediately since for k = n we get that T_n is a MST.

• Proof: We prove by induction on k. For k=1, there is only one vertex which is clearly in some MST.

T=(V,F) such that $T_k\subseteq T$. If $e_k\in F$ than $T_{k+1}\subseteq T$ By our induction hypothesis there exists a MST and we're done.

Greedy algorithms

Assume $e_k \not\in F$. It follows that the graph with edges $F \cup \{e_k\}$ has a cycle C. Let $f \in (C - e_k)$ be such that $f \in E(V_k, V - V_k)$. From the definition of e_k it follows that $w(e_k) \le w(f)$. Set $T' = (V, (F - \{f\}) \cup \{e_k\})$. It follows that T' is a tree

 $w(T') = w(T) - w(f) + w(e_k) \le w(T)$

We see that T' is a MST and $T_k\subseteq T'$. \blacksquare

Greedy algorithms

• We will require the notion of a heap.

- Informally: a binary tree is complete if all its levels are full
- \bullet If T is a complete binary tree such that each node ν has a

$$w(v) \le w(\mathsf{left}(v))$$
 and $w(v) \le w(\mathsf{right}(v))$

 \bullet For a node ν we denote by \mathcal{T}_{ν} the sub-tree whose root is

Greedy • We will require the notion of a heap.

• Informally: a binary tree is complete if all its levels are full

except maybe the last one which is "full up to some

codes
Existence of prefix codes
Shannon's theorem

point":

Minimal spanning tree Graphs (some concepts and definitions)
Minimal spanning trees Kruskal stronger trees and spanning trees Kruskal stronger trees stronger trees stronger s

• If T is a complete binary tree such that each node v has a weight value w(v), then we say that T is a heap if:

 $w(v) \le w(\operatorname{left}(v))$ and $w(v) \le w(\operatorname{right}(v))$

whenever left(ν) and right(ν) are defined.

 \bullet For a node ν we denote by T_{ν} the sub-tree whose root is

We will require the notion of a heap.

Greedy algorithms Informally: a binary tree is complete if all its levels are full except maybe the last one which is "full up to some point":



• If T is a complete binary tree such that each node ν has a weight value $w(\nu)$, then we say that T is a heap if:

$$w(v) \le w(\mathsf{left}(v))$$
 and $w(v) \le w(\mathsf{right}(v))$

whenever left(ν) and right(ν) are defined.

• For a node ν we denote by T_{ν} the sub-tree whose root is ν .

We will require the notion of a heap.

Greedy algorithms Informally: a binary tree is complete if all its levels are full except maybe the last one which is "full up to some point":

We can view an array as a complet tree.

This is an example for: $O O O A = 1_{A-1}^{A}$ The indexes satisfies:

Left(v) = 2ν , right(v) = 2ν +.

• If T is a complete binary tree such that each node v has a weight value w(v), then we say that T is a heap if:

 $w(v) \le w(\mathsf{left}(v))$ and $w(v) \le w(\mathsf{right}(v))$

whenever left(ν) and right(ν) are defined.

ullet For a node ν we denote by \mathcal{T}_{ν} the sub-tree whose root is

◇>C
時
< 面 > < 面 > < 面 > < 面 > < 面 > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a > < a >

• Assume $v \in V$ is a root of a complete tree such that $T_{\mathrm{left}(v)}$ and $T_{\mathrm{right}(v)}$ are heaps. The following algorithm turns T into a heap.

Greedy algorithms

Heapify(T)

Clearly the complexity of Heapify is O(height(T)).

>>

 If w(v) ≤ min(w(left(v)), w(right(v))) stop.
 else if w(v) > w(left(v)) and w(left(v))
 exchange w(v) and w(left(v))
 Heapify(T_{left(v)})
 else
 exchange w(v) and w(right(v))
 Heapify(T_{right(v)})
 Heapify(T_{right(v)}) $T_{\mathrm{left}(\nu)}$ and $T_{\mathrm{right}(\nu)}$ are heaps. The following algorithm turns T into a heap. ullet Assume $v \in V$ is a root of a complete tree such that exchange $w(\nu)$ and $w({\rm right}(\nu))$ Heapify $(T_{{
m right}(
u)})$ Heapify(T) Greedy algorithms

• Clearly the complexity of Heapify is O(height(T)).

 $T_{\mathrm{left}(\nu)}$ and $T_{\mathrm{right}(\nu)}$ are heaps. The following algorithm turns T into a heap. ullet Assume $v \in V$ is a root of a complete tree such that Greedy algorithms

Heapify(T)

If w(v) ≤ min(w(left(v)), w(right(v))) stop.
 If w(v) > w(left(v)) and w(left(v)) < w(right(v))
 exchange w(v) and w(left(v))
 Heapify(T_{left(v)})
 else
 exchange w(v) and w(right(v))
 Heapify(T_{right(v)})

• Clearly the complexity of Heapify is O(height(T)).

Greedy algorithms Huffman codes Existence of prefix codes Shamon is theorem Huffman algorithm

② For i = n down to 1 ② Heapify (T_i)

If A is an array (of length n) considered as a complete tree T, we can build a heap out of it by:
 MakeHeap(A,n)

Minimal

spanning tree
Graphs (some
concepts and
definitions)
Minimal
spanning trees

Spanning trees
Kruskal
algorithm for
MST
Prim's algorithm

such that $2^m \le n \le 2^{m+1} - 1$. For the complexity of MakeHeap we have: $\sum_i \operatorname{height}(T_i) = O\left(\sum_{i=1}^{2^{m+1}-1} \operatorname{height}(T_i)\right) =$

 $O\left(\sum_{j=1}^{m} (m-j)^{2^{j}}\right) = O\left(\sum_{j=1}^{m} j 2^{m-j}\right) = O(2^{m}) \sum_{j=1}^{\infty} j 2^{-j}$

Since $\sum_{i=1}^{\infty} j2^{-j}$ =

Greedy algorithms

Huffman codes Existence of prefix codes Chamoric

If A is an array (of length n) considered as a complete tree T, we can build a heap out of it by:
 MakeHeap(A,n)

• For i = n down to 1

 \bigcirc Heapify(T_i)

Minimal Spanning tre

spanning tree
Graphs (some
concepts and
definitions)
Minimal
spanning trees
Kruskal

Kruskal algorithm for MST Prim's algorithm

be such that $2^m \le n \le 2^{m+1}-1$. For the complexity of MakeHeap we have: $\sum_n \text{height}(T_i) = O\left(\frac{2^{m+1}-1}{\sum_n \text{height}(T_i)}\right) =$

 $O\left(\sum_{j=1}^{m} (m-j)2^{j}\right) = O\left(\sum_{j=1}^{m} j2^{m-j}\right) = O(2^{m}) \sum_{j=1}^{\infty} j2^{-j} = 0$

Since $\sum_{i=1}^{\infty} j2^{-j}$:

Greedy algorithms

If A is an array (of length n) considered as a complete tree T, we can build a heap out of it by:
MakeHeap(A,n)
For i = n down to 1
Heapify(T_i)

Greedy algorithms

If A is an array (of length n) considered as a complete tree T, we can build a heap out of it by:
 MakeHeap(A,n)

• For i = n down to 1 • Heapify (T_i)

 \bullet Let m be such that $2^m \le n \le 2^{m+1} - 1$. For the complexity of MakeHeap we have:

$$\sum_{j=1}^{n} \operatorname{height}(T_{j}) = O\left(2^{m+1} - 1 + \frac{1}{n}\right) = 0$$

$$O\left(\sum_{j=1}^{m} (m-j)2^{j}\right) = O\left(\sum_{j=1}^{m} j2^{m-j}\right) = O(2^{m}) \sum_{j=1}^{\infty} j2^{-j} = O(n)$$

Since $\sum_{j=1}^{\infty} j2^{-j} = 2$.

Greedy algorithms

Going back to Prim's algorithm.

• At each step of the algorithm we hold a tree $T_k = (V_k, E_k) \ (V_k = \{v_1, ..., v_k\} \ \text{and} \ E_k = \{e_1, ..., e_{k-1}\})$ and a heap whose nodes are the edges of the cut $(V_k, V - V_k)$ ordered by the weights of the edges in G.

• How do we update H_k ?

Greedy algorithms

codes
Existence of prefix codes
Shannon's theorem

Going back to Prim's algorithm.

Minimal spanning trees Graphs (some concepts and definitions) Minimal spanning trees Kruskal K

• At each step of the algorithm we hold a tree $T_k = (V_k, E_k)$ ($V_k = \{v_1, ..., v_k\}$ and $E_k = \{e_1, ..., e_{k-1}\}$) and a heap whose nodes are the edges of the cut $(V_k, V - V_k)$ ordered by the weights of the edges in G.

• Initialization: $V_k = \{v_1\}$, $E_k = \emptyset$. H_1 a heap of the edges of the cut $(V, V - v_1)$ • Step: Take the edge e = (u, v) from the root of H_k (where $u \in V_k$, $v \in V - V_k$). Define $e_k = e$, $v_{k+1} = v$ and update $V_{k+1} = V_k \cup \{v_{k+1}\}$, $E_{k+1} = E_k \cup \{e_k\}$.

• How do we update H_k ?

Greedy algorithms

Going back to Prim's algorithm.

• At each step of the algorithm we hold a tree $T_k = (V_k, E_k) (V_k = \{v_1, ..., v_k\} \text{ and } E_k = \{e_1, ..., e_{k-1}\})$ and a heap whose nodes are the edges of the cut

© Step: Take the edge e=(u,v) from the root of H_k (where $u\in V_k$, $v\in V-V_k$). Define $e_k=e, v_{k+1}=v$ and update $V_{k+1}=V_k\cup\{v_{k+1}\}, E_{k+1}=E_k\cup\{e_k\}.$ update H_k to H_{k+1} . ① Initialization: $V_k=\{v_1\},\; E_k=\emptyset.\; H_1$ a heap of the edges of the cut $(V,V-v_1)$ $(V_k, V - V_k)$ ordered by the weights of the edges in G. • How do we update H_k ?

Greedy algorithms

- Going back to Prim's algorithm.
- At each step of the algorithm we hold a tree $T_k = (V_k, E_k) (V_k = \{v_1, ..., v_k\} \text{ and } E_k = \{e_1, ..., e_{k-1}\})$ and a heap whose nodes are the edges of the cut $(V_k, V V_k)$ ordered by the weights of the edges in G.
- Initialization: $V_k = \{v_1\}$, $E_k = \emptyset$. H_1 a heap of the edges of the cut $(V, V v_1)$ Step: Take the edge e = (u, v) from the root of H_k (where $u \in V_k$, $v \in V V_k$). Define $e_k = e$, $v_{k+1} = v$ and update $V_{k+1} = V_k \cup \{v_{k+1}\}$, $E_{k+1} = E_k \cup \{e_k\}$.
 - How do we update H_k ?

Greedy algorithms

- Going back to Prim's algorithm.
- At each step of the algorithm we hold a tree $T_k = (V_k, E_k) (V_k = \{v_1, ..., v_k\} \text{ and } E_k = \{e_1, ..., e_{k-1}\})$ and a heap whose nodes are the edges of the cut $(V_k, V V_k)$ ordered by the weights of the edges in G.
- Initialization: $V_k = \{v_1\}$, $E_k = \emptyset$. H_1 a heap of the edges of the cut $(V, V v_1)$ Step: Take the edge e = (u, v) from the root of H_k (where $u \in V_k$, $v \in V V_k$). Define $e_k = e$, $v_{k+1} = v$ and update $V_{k+1} = V_k \cup \{v_{k+1}\}$, $E_{k+1} = E_k \cup \{e_k\}$.
- How do we update H_k ?

Greedy algorithms

codes
Existence of prefix codes

Huffman
algorithm
Minimal
spanning tree
Graphs (some

spanning trees
Graphs (some
concepts and
definitions)
Minimal
spanning trees
Kruskal
Kruskal
MST

When updating H_k to H_{k+1} we drop all edges of the form (x, v_{k+1}) where x ∈ V_k and we add all edges of the form (v_{k+1}, y) where y ∈ V - V_{k+1}.

Removing an edge can be done using Heapify and thus in O(log |E|).
Adding an edge can also be done in O(log |E|). (Simply add the edge as a leaf and "move it up level by level" if

necessary.)

• At each step we do $\deg(\nu_{k+1})$ removals and additions of edges so overall we do $O(\deg(\nu_{k+1})\log|E|)$ operations at

It follows that the overall complexity is:

 $O\left(\log|E|\sum_{v\in V}\deg(v)\right) = O((\log|E|)\cdot(2|E|)) = O(|E|\log|E|) = O((|E|\log|V|))$

Greedy
algorithms
Codes
Codes
Existence of perfix codes
Shamon's
Hufman
Algorithm
Alfman
Algorithm
Alfminal
Spanning trees
Graphs (some definitions)
Graphs (some definitions)
Algorithm for algorithm for algorithm for algorithm for algorithm for brin's almorithm for brin's almorithm

• When updating H_k to H_{k+1} we drop all edges of the form (x, v_{k+1}) where $x \in V_k$ and we add all edges of the form (v_{k+1}, y) where $y \in V - V_{k+1}$.

 • Removing an edge can be done using Heapify and thus in $O(\log |E|)$.

• Adding an edge can also be done in $O(\log |E|)$. (Simply add the edge as a leaf and "move it up level by level" if necessary.)

• At each step we do $\deg(v_{k+1})$ removals and additions of edges so overall we do $O(\deg(v_{k+1})\log|E|)$ operations at each step.

It follows that the overall complexity is

 $O\left(\log |E| \sum_{v \in V} \deg(v)\right) = O((\log |E|) \cdot (2|E|)) = O(|E| \log |E|) = O(|E| \log |V|)$

Greedy
algorithms
Huffman
Codes
Esistence of prefix codes
Shamon's therem
Alfordinan
algorithm
Minimal
Spanning trees
Graph's (some concepts and coffinitions)
Minimal
Minimal
Spanning trees
Graph's (some concepts and coffinitions)
Minimal
Minimal
Minimal
Spanning trees
Graph's (some concepts and coffinitions)
Minimal

• When updating H_k to H_{k+1} we drop all edges of the form (x, v_{k+1}) where $x \in V_k$ and we add all edges of the form (v_{k+1}, y) where $y \in V - V_{k+1}$.

Removing an edge can be done using Heapify and thus in O(log |E|).
Adding an edge can also be done in O(log |E|). (Simply add the edge as a leaf and "move it up level by level" if

• At each step we do $\deg(\nu_{k+1})$ removals and additions of edges so overall we do $O(\deg(\nu_{k+1})\log|E|)$ operations at each step

It follows that the overall complexity is

```
O\left(\log|E|\sum_{v\in V}\deg(v)\right)=O((\log|E|)\cdot(2|E|))=O(|E|\log|E|)=O(|E|\log|V|
```

Greedy algorithms

 \bullet When updating H_k to H_{k+1} we drop all edges of the form

Huffman codes Existence of prefix codes Minimal spanning tree Graphs (some concepts and definitions) Minimal spanning trees kruskal

(x, v_{k+1}) where x ∈ V_k and we add all edges of the form (v_{k+1}, y) where y ∈ V - V_{k+1}.
 Removing an edge can be done using Heapify and thus in O(log |E|).
 Adding an edge can also be done in O(log |E|). (Simply add the edge as a leaf and "move it up level by level" if necessary.)
 At each step we do deg(v_{k+1}) removals and additions of

 $O\left(\log|E|\sum_{v\in V}\deg(v)\right) = O((\log|E|)\cdot(2|E|)) = O(|E|\log|E|) = O(|E|\log|V|)$

• It follows that the overall complexity is:

each step.

edges so overall we do $O(\deg(\nu_{k+1})\log|E|)$ operations at

Greedy algorithms

 \bullet When updating H_k to H_{k+1} we drop all edges of the form $(x,
u_{k+1})$ where $x \in V_k$ and we add all edges of the form $(
u_{k+1},y)$ where $y\in V-V_{k+1}.$

• Removing an edge can be done using Heapify and thus in

 \bullet Adding an edge can also be done in $O(\log |E|).$ (Simply add the edge as a leaf and "move it up level by level" if $O(\log |E|)$. necessary.)

edges so overall we do $O(\deg(\nu_{k+1})\log|E|)$ operations at \bullet At each step we do $\text{deg}(\nu_{k+1})$ removals and additions of each step.

• It follows that the overall complexity is:

 $O\left(\log|E|\sum_{v\in V}\deg(v)\right) = O((\log|E|)\cdot(2|E|)) = O(|E|\log|E|) = O(|E|\log|V|)$