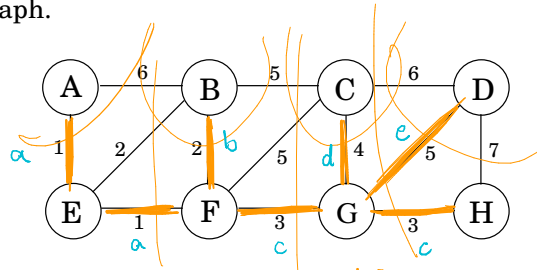# Exercises

5.1. Consider the following graph.



(a) What is the cost of its minimum spanning tree? *19*

(b) How many minimum spanning trees does it have? *2*

(c) Suppose Kruskal's algorithm is run on this graph. In what order are the edges added to the MST? For each edge in this sequence, give a cut that justifies its addition. *em azul*

5.2. Suppose we want to find the minimum spanning tree of the following graph.
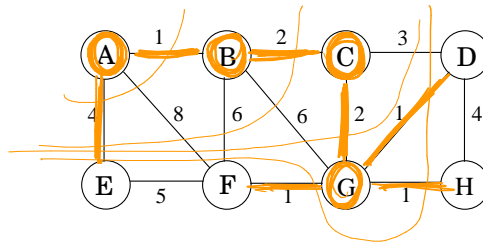


(a) Run Prim's algorithm; whenever there is a choice of nodes, always use alphabetic ordering (e.g., start from node $A$). Draw a table showing the intermediate values of the cost array. *Em laranja*

(b) Run Kruskal's algorithm on the same graph. Show how the disjoint-sets data structure looks at every intermediate stage (including the structure of the directed trees), assuming path compression is used.

5.3. Design a linear-time algorithm for the following task.

*Input:* A connected, undirected graph $G$.
*Question:* Is there an edge you can remove from $G$ while still leaving $G$ connected?

*Se a resposta for não, quantas arestas o grafo tem?*

Can you reduce the running time of your algorithm to $O(|V|)$? *DFS que para ao encontrar uma aresta p/ um vértice já visitado*

5.4. Show that if an undirected graph with $n$ vertices has $k$ connected components, then it has at least $n - k$ edges.

5.5. Consider an undirected graph $G = (V, E)$ with nonnegative edge weights $w_e \geq 0$. Suppose that you have computed a minimum spanning tree of $G$, and that you have also computed shortest paths to all nodes from a particular node $s \in V$.

Now suppose each edge weight is increased by $1$: the new weights are $w'_e = w_e + 1$.

(a) Does the minimum spanning tree change? Give an example where it changes or prove it cannot change. *Não, pois toda Árvore geradora tem n-1 arestas, tendo seu custo aumentado de n-1*

(b) Do the shortest paths change? Give an example where they change or prove they cannot change. *Provalvelmente*
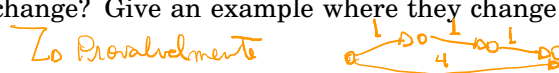
5.6. Let $G = (V, E)$ be an undirected graph. Prove that if all its edge weights are distinct, then it has a unique minimum spanning tree.

*[Handwritten top annotation:]* Na árvore de cam. mínimos $s$ está conectado diretamente ao menos a um vértice $u$. Logo
$c(s,u) = dist(s,u) \leq c(P_{su}) = \sum_{e \in P_{su}} c(e)$ $\forall$ caminho $P_{su}$ que vai de $s$ até $u$ e $c(e) \geq 0$ e distinto
$\forall e \in E$, Temos que $(s,u)$ é a aresta de menor custo que atravessa o corte $(\{s\}, V \setminus \{s\})$

**5.7.** Show how to find the *maximum* spanning tree of a graph, that is, the spanning tree of largest total weight.

**5.8.** Suppose you are given a weighted graph $G = (V, E)$ with a distinguished vertex $s$ and where all edge weights are positive and distinct. Is it possible for a tree of shortest paths from $s$ and a minimum spanning tree in $G$ to not share any edges? If so, give an example. If not, give a reason.

**5.9.** The following statements may or may not be correct. In each case, either prove it (if it is correct) or give a counterexample (if it isn't correct). Always assume that the graph $G = (V, E)$ is undirected. Do not assume that edge weights are distinct unless this is specifically stated.

*[Handwritten left margin:]* Por absurdo, tomo uma MST sem $e$, adiciono $e$, removo outra aresta do ciclo e o custo reduz. Removo $e$ da MST $T$ e vejo o custo gerado.

*[Handwritten right margin:]* $\forall$ corte separado $\forall$ ciclo, tu 2 ou mais arestas distintas corta ou Removo uma aresta de $\mu$ ciclo não desconecta o menos o por alguma aresta do ciclo formado

(a) If graph $G$ has more than $|V| - 1$ edges, and there is a unique heaviest edge, then this edge cannot be part of a minimum spanning tree. **F**

(b) If $G$ has a cycle with a unique heaviest edge $e$, then $e$ cannot be part of any MST. **V**

(c) Let $e$ be any edge of minimum weight in $G$. Then $e$ must be part of some MST. **V** *Tome uma MST e troque $e$*

(d) If the lightest edge in a graph is unique, then it must be part of every MST. **V**

(e) If $e$ is part of some MST of $G$, then it must be a lightest edge across some cut of $G$. **V**

(f) If $G$ has a cycle with a unique lightest edge $e$, then $e$ must be part of every MST. **F**

(g) The shortest-path tree computed by Dijkstra's algorithm is necessarily an MST. **F**

(h) The shortest path between two nodes is necessarily part of some MST. **F**

(i) Prim's algorithm works correctly when there are negative edges. **V** *Ela continua pegando arestas de menor custo que atravessam um corte*

(j) (For any $r > 0$, define an $r$-*path* to be a path whose edges all have weight $< r$.) If $G$ contains an $r$-path from node $s$ to $t$, then every MST of $G$ must also contain an $r$-path from node $s$ to node $t$. **V** ?

*[Handwritten left margin:]* Por absurdo, seja $P_{st} \in T$ um caminho de $s$ a $t$ na MST que não é um $r$-path?

**5.10.** Let $T$ be an MST of graph $G$. Given a connected subgraph $H$ of $G$, show that $T \cap H$ is contained in some MST of $H$. *Subestrutura ótima, provar por contradição*

**5.11.** Give the state of the disjoint-sets data structure after the following sequence of operations, starting from singleton sets $\{1\}, \ldots, \{8\}$. Use path compression. In case of ties, always make the lower numbered root point to the higher numbered one.

$$\text{union}(1,2), \text{union}(3,4), \text{union}(5,6), \text{union}(7,8), \text{union}(1,4), \text{union}(6,7), \text{union}(4,5), \text{find}(1)$$
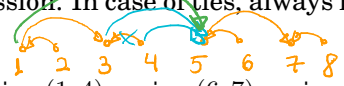
**5.12.** Suppose you implement the disjoint-sets data structure using union-by-rank but not path compression. Give a sequence of $m$ union and find operations on $n$ elements that take $\Omega(m \log n)$ time. *para $i = 1$ até $m/2k$: union$(i, i+m/2k)$, c/ $k = 1$ até $\lg n$*
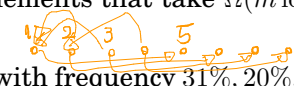
**5.13.** A long string consists of the four characters $A, C, G, T$; they appear with frequency $31\%, 20\%, 9\%$, and $40\%$, respectively. What is the Huffman encoding of these four characters?

**5.14.** Suppose the symbols $a, b, c, d, e$ occur with frequencies $1/2, 1/4, 1/8, 1/16, 1/16$, respectively.

(a) What is the Huffman encoding of the alphabet?

(b) If this encoding is applied to a file consisting of $1{,}000{,}000$ characters with the given frequencies, what is the length of the encoded file in bits?

**5.15.** We use Huffman's algorithm to obtain an encoding of alphabet $\{a, b, c\}$ with frequencies $f_a, f_b, f_c$. In each of the following cases, either give an example of frequencies $(f_a, f_b, f_c)$ that would yield the specified code, or explain why the code cannot possibly be obtained (no matter what the frequencies are).

(a) Code: $\{0, 10, 11\}$

(b) Code: $\{0, 1, 00\}$

(c) Code: $\{10, 01, 00\}$

5.16. Prove the following two properties of the Huffman encoding scheme.

(a) If some character occurs with frequency more than $2/5$, then there is guaranteed to be a codeword of length $1$.

(b) If all characters occur with frequency less than $1/3$, then there is guaranteed to be no codeword of length $1$.

5.17. Under a Huffman encoding of $n$ symbols with frequencies $f_1, f_2, \ldots, f_n$, what is the longest a codeword could possibly be? Give an example set of frequencies that would produce this case.

5.18. The following table gives the frequencies of the letters of the English language (including the blank for separating words) in a particular corpus.

| blank | 18.3% | r | 4.8% | y | 1.6% |
|---|---|---|---|---|---|
| e | 10.2% | d | 3.5% | p | 1.6% |
| t | 7.7% | l | 3.4% | b | 1.3% |
| a | 6.8% | c | 2.6% | v | 0.9% |
| o | 5.9% | u | 2.4% | k | 0.6% |
| i | 5.8% | m | 2.1% | j | 0.2% |
| n | 5.5% | w | 1.9% | x | 0.2% |
| s | 5.1% | f | 1.8% | q | 0.1% |
| h | 4.9% | g | 1.7% | z | 0.1% |

(a) What is the optimum Huffman encoding of this alphabet?

(b) What is the expected number of bits per letter?

(c) Suppose now that we calculate the entropy of these frequencies

$$H = \sum_{i=0}^{26} p_i \log \frac{1}{p_i}$$

(see the box in page 150). Would you expect it to be larger or smaller than your answer above? Explain.

(d) Do you think that this is the limit of how much English text can be compressed? What features of the English language, besides letters and their frequencies, should a better compression scheme take into account?

5.19. *Entropy.* Consider a distribution over $n$ possible outcomes, with probabilities $p_1, p_2, \ldots, p_n$.

(a) Just for this part of the problem, assume that each $p_i$ is a power of $2$ (that is, of the form $1/2^k$). Suppose a long sequence of $m$ samples is drawn from the distribution and that for all $1 \le i \le n$, the $i^{\text{th}}$ outcome occurs exactly $mp_i$ times in the sequence. Show that if Huffman encoding is applied to this sequence, the resulting encoding will have length

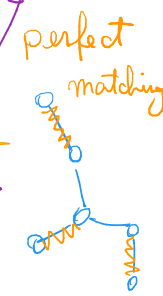$$\sum_{i=1}^{n} mp_i \log \frac{1}{p_i}.$$

157

(b) Now consider arbitrary distributions—that is, the probabilities $p_i$ are not restricted to powers of 2. The most commonly used measure of the *amount of randomness* in the distribution is the *entropy*

$$\sum_{i=1}^{n} p_i \log \frac{1}{p_i}.$$

For what distribution (over $n$ outcomes) is the entropy the largest possible? The smallest possible?

*no perfect matching*

*perfect matching*

→ Tome uma folha $u$, coloque sua única aresta $\{u,v\}$ no matching $M$, remova $u$ e $v$ do grafo, repita

5.20. Give a linear-time algorithm that takes as input a tree and determines whether it has a *perfect matching*: a set of edges that touches each node exactly once.

**5.20'** A *feedback edge set* of an undirected graph $G = (V, E)$ is a subset of edges $E' \subseteq E$ that intersects every cycle of the graph. Thus, removing the edges $E'$ will render the graph acyclic.

Give an efficient algorithm for the following problem:

*Input:* Undirected graph $G = (V, E)$ with positive edge weights $w_e$

*Output:* A feedback edge set $E' \subseteq E$ of minimum total weight $\sum_{e \in E'} w_e$

→ O contrário uma Max ST T e devolva $E(G) \setminus T$

5.21. In this problem, we will develop a new algorithm for finding minimum spanning trees. It is based upon the following property:

Pick any cycle in the graph, and let $e$ be the heaviest edge in that cycle. Then there is a minimum spanning tree that does not contain $e$.

(a) Prove this property carefully. *Remova a aresta pesada da suposta MST. Alguma aresta do ciclo conecta as partes*

(b) Here is the new MST algorithm. The input is some undirected graph $G = (V, E)$ (in adjacency list format) with edge weights $\{w_e\}$.

*Na a MST Te custo menor. Absurdo*

```
sort the edges according to their weights
for each edge e ∈ E, in decreasing order of w_e:
    if e is part of a cycle of G:
        G = G - e (that is, remove e from G)
return G
```

Prove that this algorithm is correct.

(c) On each iteration, the algorithm must check whether there is a cycle containing a specific edge $e$. Give a linear-time algorithm for this task, and justify its correctness. *BFS ou DFS*

(d) What is the overall time taken by this algorithm, in terms of $|E|$? Explain your answer. $|E|^2$

5.22. You are given a graph $G = (V, E)$ with positive edge weights, and a minimum spanning tree $T = (V, E')$ with respect to these weights; you may assume $G$ and $T$ are given as adjacency lists. Now suppose the weight of a particular edge $e \in E$ is modified from $w(e)$ to a new value $\hat{w}(e)$. You wish to quickly update the minimum spanning tree $T$ to reflect this change, without recomputing the entire tree from scratch. There are four cases. In each case give a linear-time algorithm for updating the tree.

(a) $e \notin E'$ and $\hat{w}(e) > w(e)$. *Nada a fazer. Contínua existindo corte em que e é mínima*

(b) $e \notin E'$ and $\hat{w}(e) < w(e)$. *$e = \{u,v\}$. Busca de $u$ até $v$ em T. Se $e$ custar menos que alguma aresta do caminho troque*

(c) $e \in E'$ and $\hat{w}(e) < w(e)$. *Nada a fazer. Contínua existindo corte em que e é mínima*

(d) $e \in E'$ and $\hat{w}(e) > w(e)$. *$e = \{u,v\}$. Busca em $T \setminus \{e\}$ a partir de $u$. Coloque em $T \setminus \{e\}$ a aresta mais barato que atravessa o corte $(T_u, T_v)$*

5.23. Sometimes we want light spanning trees with certain special properties. Here's an example.

*Input:* Undirected graph $G = (V, E)$; edge weights $w_e$; subset of vertices $U \subset V$

*Output:* The lightest spanning tree in which the nodes of $U$ are *leaves* (there might be other leaves in this tree as well).

(The answer isn't necessarily a minimum spanning tree.)

Give an algorithm for this problem which runs in $O(|E| \log |V|)$ time. (*Hint:* When you remove nodes $U$ from the optimal solution, what is left?) *Renova U de V, encotra una MST (se estiman conexo), conecte cada v e U c/a aresta de meno custo*

5.24. A binary counter of unspecified length supports two operations: `increment` (which increases its value by one) and `reset` (which sets its value back to zero). Show that, starting from an initially zero counter, any sequence of $n$ `increment` and `reset` operations takes time $O(n)$; that is, the amortized time per operation is $O(1)$.

5.25. Here's a problem that occurs in automatic program analysis. For a set of variables $x_1, \ldots, x_n$, you are given some *equality* constraints, of the form "$x_i = x_j$" and some *disequality* constraints, of the form "$x_i \neq x_j$." Is it possible to satisfy all of them?

For instance, the constraints *Cada variável $x_i$ corresponde a um vértice $v_i$. Para cada igualdade $x_i = x_j$ temos uma aresta $\{v_i, v_j\}$ (ou um union(i,j)). Por fim, verificam, pr cada desigualdade $x_i \neq x_j$ se $v_i$ e $v_j$ estão em componentes distintos.*

$$x_1 = x_2, x_2 = x_3, x_3 = x_4, x_1 \neq x_4$$

cannot be satisfied. Give an efficient algorithm that takes as input $m$ constraints over $n$ variables and decides whether the constraints can be satisfied.

5.26. *Graphs with prescribed degree sequences.* Given a list of $n$ positive integers $d_1, d_2, \ldots, d_n$, we want to efficiently determine whether there exists an undirected graph $G = (V, E)$ whose nodes have degrees precisely $d_1, d_2, \ldots, d_n$. That is, if $V = \{v_1, \ldots, v_n\}$, then the degree of $v_i$ should be exactly $d_i$. We call $(d_1, \ldots, d_n)$ the *degree sequence of $G$*. This graph $G$ should not contain self-loops (edges with both endpoints equal to the same node) or multiple edges between the same pair of nodes.

(a) Give an example of $d_1, d_2, d_3, d_4$ where all the $d_i \leq 3$ and $d_1 + d_2 + d_3 + d_4$ is even, but for which no graph with degree sequence $(d_1, d_2, d_3, d_4)$ exists.

(b) Suppose that $d_1 \geq d_2 \geq \cdots \geq d_n$ and that there exists a graph $G = (V, E)$ with degree sequence $(d_1, \ldots, d_n)$. We want to show that there must exist a graph that has this degree sequence and where in addition the neighbors of $v_1$ are $v_2, v_3, \ldots, v_{d_1+1}$. The idea is to gradually transform $G$ into a graph with the desired additional property.

   i. Suppose the neighbors of $v_1$ in $G$ are not $v_2, v_3, \ldots, v_{d_1+1}$. Show that there exists $i < j \leq n$ and $u \in V$ such that $\{v_1, v_i\}, \{u, v_j\} \notin E$ and $\{v_1, v_j\}, \{u, v_i\} \in E$.

   ii. Specify the changes you would make to $G$ to obtain a new graph $G' = (V, E')$ with the same degree sequence as $G$ and where $(v_1, v_i) \in E'$.

   iii. Now show that there must be a graph with the given degree sequence but in which $v_1$ has neighbors $v_2, v_3, \ldots, v_{d_1+1}$.

(c) Using the result from part (b), describe an algorithm that on input $d_1, \ldots, d_n$ (not necessarily sorted) decides whether there exists a graph with this degree sequence. Your algorithm should run in time polynomial in $n$ and in $m = \sum_{i=1}^{n} d_i$.

5.27. Alice wants to throw a party and is deciding whom to call. She has $n$ people to choose from, and she has made up a list of which pairs of these people know each other. She wants to pick as many people as possible, subject to two constraints: at the party, each person should have at least five other people whom they know *and* five other people whom they don't know.

Give an efficient algorithm that takes as input the list of $n$ people and the list of pairs who know each other and outputs the best choice of party invitees. Give the running time in terms of $n$.

5.28. A *prefix-free encoding* of a finite alphabet $\Gamma$ assigns each symbol in $\Gamma$ a binary codeword, such that no codeword is a prefix of another codeword.

Show that such an encoding can be represented by a full binary tree in which each leaf corresponds to a unique element of $\Gamma$, whose codeword is generated by the path from the root to that leaf (interpreting a left branch as $0$ and a right branch as $1$).

5.29. *Ternary Huffman.* Trimedia Disks Inc. has developed "ternary" hard disks. Each cell on a disk can now store values $0$, $1$, or $2$ (instead of just $0$ or $1$). To take advantage of this new technology, provide a modified Huffman algorithm for compressing sequences of characters from an alphabet of size $n$, where the characters occur with known frequencies $f_1, f_2, \ldots, f_n$. Your algorithm should encode each character with a variable-length codeword over the values $0, 1, 2$ such that no codeword is a prefix of another codeword and so as to obtain the maximum possible compression. Prove that your algorithm is correct.

5.30. The basic intuition behind Huffman's algorithm, that frequent blocks should have short encodings and infrequent blocks should have long encodings, is also at work in English, where typical words like I, you, is, and, to, from, and so on are short, and rarely used words like velociraptor are longer.

However, words like fire!, help!, and run! are short not because they are frequent, but perhaps because time is precious in situations where they are used.

To make things theoretical, suppose we have a file composed of $m$ different words, with frequencies $f_1, \ldots, f_m$. Suppose also that for the $i$th word, the cost per bit of encoding is $c_i$. Thus, if we find a prefix-free code where the $i$th word has a codeword of length $l_i$, then the total cost of the encoding will be $\sum_i f_i \cdot c_i \cdot l_i$.

Show how to modify Huffman's algorithm to find the prefix-free encoding of minimum total cost.

5.31. A server has $n$ customers waiting to be served. The service time required by each customer is known in advance: it is $t_i$ minutes for customer $i$. So if, for example, the customers are served in order of increasing $i$, then the $i$th customer has to wait $\sum_{j=1}^{i} t_j$ minutes.

We wish to minimize the total waiting time

$$T = \sum_{i=1}^{n} (\text{time spent waiting by customer } i).$$

Give an efficient algorithm for computing the optimal order in which to process the customers.

5.32. Show how to implement the stingy algorithm for Horn formula satisfiability (Section 5.3) in time that is linear in the length of the formula (the number of occurrences of literals in it). (*Hint:* Use a directed graph, with one node per variable, to represent the implications.)

5.33. Show that for any integer $n$ that is a power of $2$, there is an instance of the set cover problem (Section 5.4) with the following properties:

   i. There are $n$ elements in the base set.

   ii. The optimal cover uses just two sets.

   iii. The greedy algorithm picks at least $\log n$ sets.

Thus the approximation ratio we derived in the chapter is tight.