Chapter 4
Greedy Algorithm and Spanning Tree

Greed, in the end, fails even the greedy.

—Cathryn Louis

Self-reducibility is the backbone of each greedy algorithm in which self-reducibility structure is a tree of special kind, i.e., its internal nodes lie on a path. In this chapter, we study algorithms with such a self-reducibility structure and related combinatorial theory supporting greedy algorithms.

4.1 Greedy Algorithms

A problem that the greedy algorithm works for computing optimal solutions often has the self-reducibility and a simple exchange property. Let us use two examples to explain this point.

Example 4.1.1 (Activity Selection) Consider n activities with starting times $s_1, s_2, \ldots, s_n$ and ending times $f_1, f_2, \ldots, f_n$, respectively. They may be represented by intervals $[s_1, f_1)$, $[s_2, f_2)$, …, and $[s_n, f_n)$. The problem is to find a maximum subset of nonoverlapping activities, i.e., nonoverlapping intervals.

This problem has the following exchange property.

Lemma 4.1.2 (Exchange Property) *Suppose $f_1 \leq f_2 \leq \cdots \leq f_n$. In a maximum solution without interval $[s_1, f_1)$, we can always exchange $[s_1, f_1)$ with the first activity in the maximum solution preserving the maximality.*

Proof Let $[s_i, f_i)$ be the first activity in the maximum solution mentioned in the lemma. Since $f_1 \leq f_i$, replacing $[s_i, f_i)$ by $[s_1, f_1)$ will not cost any overlapping. □

The following lemma states a self-reducibility.

D.-Z. Du et al., *Introduction to Combinatorial Optimization*, Springer Optimization and Its Applications 196, https://doi.org/10.1007/978-3-031-10596-8_4

Lemma 4.1.3 (Self-Reducibility) *Suppose* $\{I_1^*, I_2^*, \ldots, I_k^*\}$ *is an optimal solution. Then,* $\{I_2^*, \ldots, I_k^*\}$ *is an optimal solution for the activity problem on input* $\{I_i \mid I_i \cap I_1^*\}$ *where* $I_i = [s_i, f_i)$.

Proof For contradiction, suppose that $\{I_2^*, \ldots, I_k^*\}$ is not an optimal solution for the activity problem on input $\{I_i \mid I_i \cap I_1^*\}$. Then, $\{I_i \mid I_i \cap I_1^*\}$ contains k nonoverlapping activities, which all are not overlapping with I_1^*. Putting I_1^* in these k activities, we will obtain a feasible solution containing $k + 1$ activities, contradicting the assumption that $\{I_1^*, I_2^*, \ldots, I_k^*\}$ is an optimal solution. □

Based on Lemmas 4.1.2 and 4.1.3, we can design a greedy algorithm in Algorithm 11 and obtain the following result.

Algorithm 11 Greedy algorithm for activity selection

Input: A sequence of n activities $[s_1, f_1), [s_2, f_2), \ldots, [s_n, f_n)$.
Output: A maximum subset of nonoverlapping activities.

sort all activities into ordering $f_1 \le f_2 \le \ldots \le f_n$
$S \leftarrow \emptyset$
for $i \leftarrow 1$ to n **do**
  if $[s_i, f_i)$ does not overlap any activity in S **then**
    $S \leftarrow S \cup \{[s_i, f_i)\}$
  end if
end for
return S

Theorem 4.1.4 *Algorithm 11 produces an optimal solution for the activity selection problem.*

Proof Let us prove it by induction on n. For $n = 1$, it is trivial.

Consider $n \ge 2$. Suppose $\{I_1^*, I_2^*, \ldots, I_k^*\}$ is an optimal solution. By Lemma 4.1.2, we may assume that $I_1^* = [s_1, f_1)$. By Lemma 4.1.3, $\{I_2^*, \ldots, I_k^*\}$ is an optimal solution for the activity selection problem on input $\{I_i \mid I_i \cap I_1^* = \emptyset\}$.

Note that after select $[s_1, f_1)$, if we ignore all iterations i with $[s_i, f_i) \cap [s_1, f_1) \neq \emptyset$, then the remaining part is the same as greedy algorithm running on input $\{I_i \mid I_i \cap I_1^* = \emptyset\}$. By induction hypothesis, it will produce an optimal solution for the activity selection problem on input $\{I_i \mid I_i \cap I_1^* = \emptyset\}$, which must contain $k - 1$ activities. Together with $[s_1, f_1)$, they form a subset of k non-overlapping activities, which should be optimal. □

Next, we study another example.

Example 4.1.5 (Huffman Tree) Given n characters $a_1, a_2, \ldots, a_n$ with weights $f_1, f_2, \ldots, f_n$, respectively, find a binary tree with n leaves labeled by $a_1, a_2, \ldots, a_n$, respectively, to minimize

$$d(a_1) \cdot f_1 + d(a_2) \cdot f_2 + \cdots + d(a_n) \cdot f_n$$

where $d(a_i)$ is the depth of leaf a_i, i.e., the number of edges on the path from the root to a_i.

First, we show a property of optimal solutions.

Lemma 4.1.6 *In any optimal solution, every internal node has two children, i.e., every optimal binary tree is full.*

Proof If an internal node has only one child, then this internal node can be removed to reduce the objective function value. □

We can also show an exchange property and a self-reducibility.

Lemma 4.1.7 (Exchange Property) *If $f_i > f_j$ and $d(a_i) > d(a_j)$, then exchanging a_i with a_j would make the objective function value decrease.*

Proof Let $d'(a_i)$ and $d(a_j)$ be the depths of a_i and a_j, respectively, after exchanging a_i with a_j. Then $d'(a_i) = d(a_j)$ and $d'(a_j) = d(a_i)$. Therefore, the difference of objective function values before and after exchange is

$$\begin{aligned}
&(d(a_i) \cdot f_i + d(a_j) \cdot f_j) - (d'(a_i) \cdot f_i + d'(a_j) \cdot f_j)\\
&\quad = (d(a_i) \cdot f_i + d(a_j) \cdot f_j) - (d(a_j) \cdot f_i + d(a_i) \cdot f_j)\\
&\quad = (d(a_i) - d(a_j))(f_i - f_j)\\
&\quad > 0
\end{aligned}$$

□

Lemma 4.1.8 (Self-Reducibility) *In any optimal tree T^*, if we assign the weight of an internal node u with the total weight w_u of its descendant leaves, then removal of the subtree T_u at the internal node results in an optimal tree T'_u for weights at remainder's leaves (Fig. 4.1).*

Proof Let $c(T)$ denote the objective function value of tree T, i.e.,

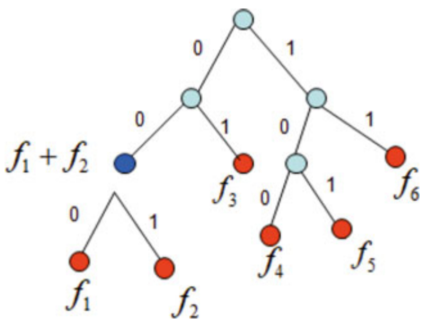

Fig. 4.1 A self-reducibility

$$c(T) = \sum_{a \text{ over leaves of } T} d(a) \cdot f(a)$$

where $d(a)$ is the depth of leaf a and $f(a)$ is the weight of leaf a. Then we have

$$c(T^*) = c(T_u) + c(T'_u).$$

If T'_u is not optimal for weights at leaves of T'_u, then we have a binary tree T''_u for those weights with $c(T''_u) < c(T'_u)$. Therefore, $c(T_u \cup T''_u) < c(T^*)$, contradicting optimality of T^*. □

By Lemmas 4.1.7 and 4.1.8, we can construct an optimal Huffman tree in the following:

- Sort $f_1 \leq f_2 \leq \cdots \leq f_n$.
- By exchange property, there must exist an optimal tree in which a_1 and a_2 are sibling at bottom level.
- By self-reducibility, the problem can be reduced to construct optimal tree for leaves weights $\{f_1 + f_2, f_3, \ldots, f_n\}$.
- Go back to initial sorting step. This process continues until only two weights exist.

In Fig. 4.2, an example is presented to explain this construction. This construction can be implemented with min-priority queue (Algorithm 12)

The Huffman tree problem is raised from the study of Huffman codes as follows.

Problem 4.1.9 (Huffman Codes) Given n characters $a_1, a_2, \ldots, a_n$ with frequencies $f_1, f_2, \ldots, f_n$, respectively, find prefix binary codes $c_1, c_2, \ldots, c_n$ to minimize

$$|c_1| \cdot f_1 + |c_2| \cdot f_2 + \cdots + |c_n| \cdot f_n,$$

where $|c_i|$ is the length of code c_i, i.e., the number of symbols in c_i.

Actually, $c_1, c_2, \ldots, c_n$ are called *prefix* binary codes if no one is a prefix of another one. Therefore, they have a binary tree representation.

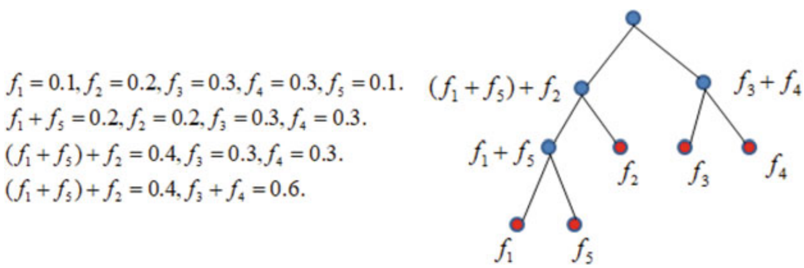

Fig. 4.2 An example for construction of Huffman tree

Algorithm 12 Greedy algorithm for Huffman tree

Input: A sequence of leaf weights $\{f_1, f_2, \ldots, f_n\}$.
Output: A binary tree.

```
Put f_1, f_2, ..., f_n into a min-priority queue Q
for i ← 1 to n − 1 do
    allocate a new node z
    left[z] ← x ← Extract-Min(Q)
    right[z] ← y ← Extract-Min(Q)
    f[z] ← f[x] + f[y]
    Insert(Q, z)
end for
return Extract-Min(Q)
```

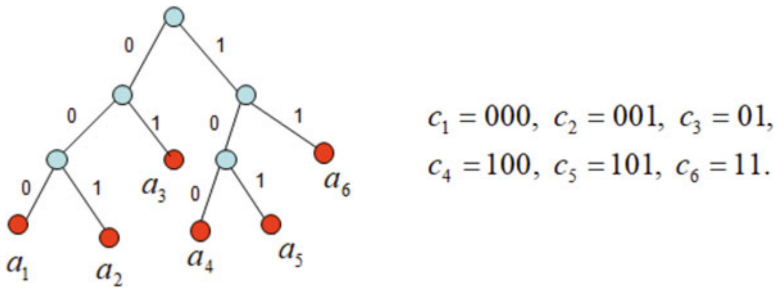

Fig. 4.3 Huffman codes

- Each edge is labeled with 0 or 1.
- Each code is represented by a path from the root to a leaf.
- Each leaf is labeled with a character.
- The length of a code is the length of corresponding path.

An example is as shown in Fig. 4.3. With this representation, the Huffman codes problem can be transformed exactly to the Huffman tree problem.

In Chap. 1, we see that the Kruskal greedy algorithm can compute the minimum spanning tree. Thus, we may have a question: Does the minimum spanning tree problem have an exchange property and self-reducibility? The answer is yes, and they are given in the following.

Lemma 4.1.10 (Exchange Property) *For an edge e with the smallest weight in a graph G and a minimum spanning tree T without e, there must exist an edge e' in T such that $(T \setminus e') \cup e$ is still a minimum spanning tree.*

Proof Suppose u and v are two endpoints of edge e. Then T contains a path p connecting u and v. On path p, every edge e' must have weight $c(e') = c(e)$. Otherwise, $(T \setminus e') \cup e$ will be a spanning tree with total weight smaller than $c(T)$, contradicting minimality of $c(T)$.

Now, select any edge e' in path p. Then $(T \setminus e') \cup e$ is a minimum spanning tree. □

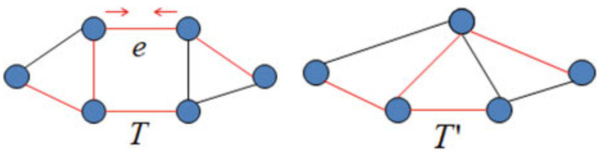

Fig. 4.4 Lemma 4.1.11

Lemma 4.1.11 (Self-Reducibility) *Suppose T is a minimum spanning tree of a graph G and edge e in T has the smallest weight. Let G' and T' be obtained from G and T, respectively, by shrinking e into a node (Fig. 4.4). Then T' is a minimum spanning tree of G'.*

Proof Note that T is a minimum spanning tree of G if and only if T' is a minimum spanning tree of G'. □

With the above two lemmas, we are able to give an alternative proof for correctness of the Kruskal algorithm. We leave it as an exercise for readers.

4.2 Matroid

There is a combinatorial structure which has a close relationship with greedy algorithms. This is the matroid. To introduce matroid, let us first study independent systems.

Consider a finite set S and a collection $\mathcal{C}$ of subsets of S. $(S, \mathcal{C})$ is called an *independent system* if

$$A \subset B, B \in \mathcal{C} \Rightarrow A \in \mathcal{C},$$

i.e., it is *hereditary*. In the independent system $(S, \mathcal{C})$, each subset in $\mathcal{C}$ is called an independent set.

Consider a maximization problem as follows.

Problem 4.2.1 (Independent Set Maximization) Let c be a nonnegative cost function on S. Denote $c(A) = \sum_{x \in A} c(x)$ for any $A \subseteq S$. The problem is to maximize $c(A)$ subject to $A \in \mathcal{C}$.

Also, consider the greedy algorithm in Algorithm 13.

For any $F \subseteq E$, a subset I of F is called a *maximal* independent subset if no independent subset of E contains F as a proper subset. Define

$$u(F) = \max\{|I| \mid I \text{ is an independent subset of } F\},$$

$$v(F) = \min\{|I| \mid I \text{ is a maximal independent subset of } F\}.$$

Algorithm 13 Greedy algorithm for independent set maximization

Input: An independent system $(S, \mathcal{C})$ with a nonnegative cost function c on S.
Output: An independent set.

Sort all elements in S into ordering $c(x_1) \geq c(x_2) \geq \cdots \geq c(x_n)$
$A \leftarrow \emptyset$
for $i \leftarrow 1$ to n **do**
  if $A \cup \{x_i\} \in \mathcal{C}$ **then**
    $A \leftarrow A \cup \{x_i\}$
  end if
end for
return A

where $|I|$ is the number of elements in I. Then we have the following theorem to estimate the performance of Algorithm 13.

Theorem 4.2.2 *Let A_G be a solution obtained by Algorithm 13. Let A^* be an optimal solution for the independent set maximization. Then*

$$1 \leq \frac{c(A^*)}{c(A_G)} \leq \max_{F \subseteq S} \frac{u(F)}{v(F)}.$$

Proof Note that $S = \{x_1, x_2, \ldots, x_n\}$ and $c(x_1) \geq c(x_2) \geq \cdots \geq c(x_n)$. Denote $S_i = \{x_1, \ldots, x_i\}$. Then

$$\begin{aligned} c(A_G) &= c(x_1)|S_1 \cap A_G| + \sum_{i=2}^{n} c(x_i)(|S_i \cap A_G| - |A_{i-1} \cap A_G|) \\ &= \sum_{i=1}^{n-1} |S_i \cap A_G|(c(x_i) - c(x_{i+1})) + |A_n \cap A_G| c(x_n). \end{aligned}$$

Similarly,

$$c(A^*) = \sum_{i=1}^{n-1} |S_i \cap A^*|(c(x_i) - c(x_{i+1})) + |S_n \cap A^*| c(x_n).$$

Thus,

$$\frac{c(A^*)}{c(A_G)} \leq \max_{1 \leq i \leq n} \frac{|A^* \cap S_i|}{|A_G \cap S_i|}.$$

We claim that $A_i \cap A_G$ is a maximal independent subset of S_i. In fact, for contradiction, suppose that $S_i \cap A_G$ is not a maximal independent subset of S_i. Then there exists an element $x_j \in S_i \setminus A_G$ such that $(S_i \cap A_G) \cup \{x_j\}$ is independent.

Thus, in the computation of Algorithm 2.1, $I \cup \{e_j\}$ as a subset of $(S_i \cap A_G)\{x_j\}$ should be independent. This implies that x_j should be in A_G, a contradiction.

Now, from our claim, we see that

$$|S_i \cap A_G| \geq v(S_i).$$

Moreover, since $S_i \cap A^*$ is independent, we have

$$|S_i \cap A^*| \leq u(S_i).$$

Therefore,

$$\frac{c(A^*)}{c(A_G)} \leq \max_{F \subseteq S} \frac{u(F)}{v(F)}.$$

□

The matroid is an independent system satisfying an additional property, called *augmentation property*:

$$A, B \in \mathcal{C} \text{ and } |A| > |B|$$
$$\Rightarrow \exists x \in A \setminus B : B \cup \{x\} \in \mathcal{C}.$$

This property is equivalent to some others.

Theorem 4.2.3 *An independent system $(S, \mathcal{C})$ is a matroid if and only if for any $F \subseteq S$, $u(F) = v(F)$.*

Proof For forward direction, consider two maximal independent sets A and B. If $|A| > |B|$, then there exists $x \in A \setminus B$ such that $B \cup \{x\} \in \mathcal{C}$, contradicting maximality of B.

For backward direction, consider two independent sets with $|A| > |B|$. Set $F = A \cup B$. Then every maximal independent set of F has size at least $|A|$ ($> |B|$). Hence, B cannot be a maximal independent set of F. Thus, there exists an element $x \in F \setminus B = A \setminus B$ such that $B \cup \{x\} \in \mathcal{C}$. □

Theorem 4.2.4 *An independent system $(S, \mathcal{C})$ is a matroid if and only if for any cost function $c(\cdot)$, Algorithm 13 gives a maximum solution.*

Proof For necessity, we note that when $(S, \mathcal{C})$ is matroid, we have $u(F) = v(F)$ for any $F \subseteq S$. Therefore, Algorithm 13 gives an optimal solution.

For sufficiency, we give a contradiction argument. To this end, suppose independent system $(S, \mathcal{C})$ is not a matroid. Then, there exists $F \subseteq S$ such that F has two maximal independent sets I and J with $|I| < |J|$. Define

$$c(e) = \begin{cases} 1+\varepsilon & \text{if } e \in I \\ 1 & \text{if } e \in J \setminus I \\ 0 & \text{otherwise} \end{cases}$$

where ε is a sufficient small positive number to satisfy $c(I) < c(J)$. The greedy algorithm will produce I, which is not optimal. □

This theorem gives tight relationship between matroids and greedy algorithms, which is built up on all nonnegative objective function. It may be worth mentioning that the greedy algorithm reaches optimal for a certain class of objective functions may not provide any additional information to the independent system. The following is a counterexample.

Example 4.2.5 Consider a complete bipartite graph $G = (V_1, V_2, E)$ with $|V_1| = |V_2|$. Let $\mathcal{I}$ be the family of all matchings. Clearly, $(E, \mathcal{I})$ is an independent system. However, it is not a matroid. An interesting fact is that maximal matchings may have different cardinalities for some subgraph of G although all maximal matchings for G have the same cardinality.

Furthermore, consider the problem $\max\{c(\cdot) \mid I \in \mathcal{I}\}$, called the *maximum assignment* problem.

If $c(\cdot)$ is a nonnegative function such that for any $u, u' \in V_1$ and $v, v' \in V_2$,

$$c(u, v) \geq \max(c(u, v'), c(u', v)) \Longrightarrow c(u, v) + c(u', v') \geq c(u, v') + c(u', v).$$

This means that replacing edges (u_1, v') and (u', v_1) in M^* by (u_1, v_1) and (u', v') will not decrease the total cost of the matching. Similarly, we can put all (u_i, v_i) into an optimal solution, that is, they form an optimal solution. This gives an exchange property. Actually, we can design a greedy algorithm to solve the maximum assignment problem. (We leave this as an exercise.)

Next, let us present some examples of the matroid.

Example 4.2.6 (Linear Vector Space) Let S be a finite set of vectors and $\mathcal{I}$ the family of linearly independent subsets of S. Then $(S, \mathcal{I})$ is a matroid.

Example 4.2.7 (Graph Matroid) Given a graph $G = (V, E)$ where V and E are its vertex set and edge set, respectively. Let $\mathcal{I}$ be the family of edge sets of acyclic subgraphs of G. Then $(E, \mathcal{I})$ is a matroid.

Proof Clearly, $(E, \mathcal{I})$ is an independent system. Consider a subset F of E. Suppose that the subgraph (V, F) has m connected components. Note that in each connected component, every maximal acyclic subgraph must be a spanning tree which has the number of edges one less than the number of vertices. Thus, every maximal acyclic subgraph of (V, E) has exactly $|V| - m$ edges. By Theorem 4.2.3, $(E, \mathcal{I})$ is a matroid. □

In a matroid, all maximal independent subsets have the same cardinality. They are also called *bases*. In a graph matroid obtained from a connected graph, every base is a spanning tree.

Let $\mathcal{B}$ be the family of all bases of a matroid $(S, \mathcal{C})$. Consider the following problem:

Problem 4.2.8 (Base Cost Minimization) Consider a matroid $(S, \mathcal{C})$ with base family $\mathcal{B}$ and a nonnegative cost function on S. The problem is to minimize $c(B)$ subject to $B \in \mathcal{B}$.

Algorithm 14 Greedy algorithm for base cost minimization

Input: A matroid $(S, \mathcal{C})$ with a nonnegative cost function c on S.
Output: A base.

```
1: Sort all elements in S into ordering c(x_1) ≤ c(x_2) ≤ ··· ≤ c(x_n)
2: A ← ∅
3: for i ← 1 to n do
4:     if A ∪ {x_i} ∈ C then
5:         A ← A ∪ {x_i}
6:     end if
7: end for
8: return A
```

Theorem 4.2.9 *An optimal solution of the base cost minimization can be computed by Algorithm 14, a variation of Algorithm 13.*

Proof Suppose that every base has the cardinality m. Let M be a positive number such that for any $e \in S$, $c(e) < M$. Define $c'(e) = M - c(e)$ for all $e \in E$. Then $c'(\cdot)$ is a positive function on S, and the non-decreasing ordering with respect to $c(\cdot)$ is the non-increasing ordering with respect to $c'(\cdot)$. Note that $c'(B) = mM - c(B)$ for any $B \in \mathcal{B}$. Since Algorithm 13 produces a base with maximum value of c', Algorithm 14 produces a base with minimum value of function c. □

The correctness of greedy algorithm for the minimum spanning tree can also be obtained from this theorem.

Next, consider the following problem.

Problem 4.2.10 (Unit-Time Task Scheduling) Consider a set of n unit-time tasks, $S = \{1, 2, \ldots, n\}$. Each task i can be processed during a unit-time and has to be completed before an integer deadline d_i and, if not completed, will receive a penalty w_i. The problem is to find a schedule for S on a machine within time n to minimize total penalty.

A set of tasks is independent if there exists a schedule for these tasks without penalty. Then we have the following.

Lemma 4.2.11 *A set A of tasks is independent if and only if for any $t = 1, 2, \ldots, n$, $N_t(A) \leq t$ where $N_t(A) = |\{i \in A \mid d_i \leq t\}|$.*

Proof It is trivial for "only if" part. For the "if" part, note that if the condition holds, then tasks in A can be scheduled in order of nondecreasing deadlines without penalty. □

Example 4.2.12 Let S be a set of unit-time tasks with deadlines and penalties and $\mathcal{C}$ the collection of all independent subsets of S. Then, $(S, \mathcal{C})$ is a matroid. Therefore, an optimal solution for the unit-time task scheduling problem can be computed by a greedy algorithm (i.e., Algorithm 13).

Proof (Hereditary) Trivial.

(Augmentation) Consider two independent sets A and B with $|A| < |B|$. Let k be the largest k such that $N_t(A) \geq N_t(B)$. (A few examples are presented in Fig. 4.5 to explain the definition of k.) Then $k < n$ and $N_t(A) < N_t(B)$ for $k+1 \leq t \leq n$. Choose $x \in \{i \in B \setminus A \mid d_i = k+1\}$. Then

$$N_t(A \cup \{x\}) = N_t(A) \leq t \text{ for } 1 \leq t \leq k$$

and

$$N_t(A \cup \{x\}) \leq N_t(A) + 1 \leq N_t(B) \leq t \text{ for } k+1 \leq t \leq n.$$

□

Example 4.2.13 Consider an independent system $(S, \mathcal{C})$. For any fixed $A \subseteq S$, define

$$\mathcal{C}_A = \{B \subseteq S \mid A \not\subseteq B\}.$$

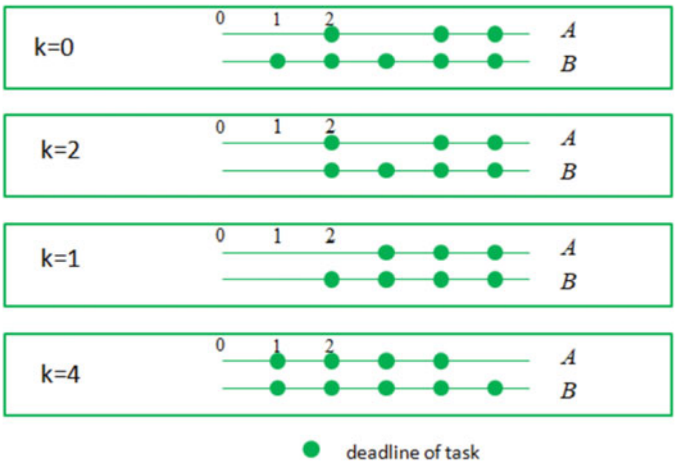

Fig. 4.5 In proof of Example 4.2.12

Then, $(S, \mathcal{C}_A)$ is a matroid.

Proof Consider any $F \subseteq S$. If $A \not\subseteq F$, then F has unique maximal independent set, which is F. Hence, $u(F) = v(F)$.

If $A \subseteq F$, then every maximal independent subset of F is in the form $F \setminus \{x\}$ for some $x \in A$. Hence, $u(F) = v(F) = |F| - 1$. □

4.3 Minimum Spanning Tree

Let us revisit the minimum spanning tree problem.

Consider a graph $G = (V, E)$ with nonnegative edge weight $c : E \to R_+$, and a spanning tree T. Let (u, v) be an edge in T. Removal (u, v) would break T into two connected components. Let U and W be vertex sets of these two components, respectively. The edges between U and V constitute a *cut*, denoted by (U, W). The cut (U, W) is said to be induced by deleting (u, v). For example, in Fig. 4.6, deleting $(3, 4)$ induces a cut $(\{1, 2, 3\}, \{4, 5, 6, 7, 8\})$.

Theorem 4.3.1 (Cut Optimality) *A spanning tree T^* is a minimum spanning tree if and only if it satisfies the cut optimality condition as follows:*

Cut Optimality Condition *For every edge (u, v) in T^*, $c(u, v) \leq c(x, y)$ for every edge (x, y) contained in the cut induced by deleting (u, v).*

Proof Suppose, for contradiction, that $c(u, v) > c(x, y)$ for some edge (x, y) in the cut induced by deleting (u, v) from T^*. Then $T' = (T^* \setminus (u, v)) \cup (x, y)$ is a spanning tree with cost less than $c(T^*)$, contradicting the minimality of T^*.

Conversely, suppose that T^* satisfies the cut optimality condition. Let T' be a minimum spanning tree such that among all minimum spanning trees, T' is the one with the most edges in common with T^*. Suppose, for contradiction, that $T' \neq T^*$. Consider an edge (u, v) in $T^* \setminus T'$. Let p be the path from u to v in T'. Then p has at least one edge (x, y) in the cut induced by deleting (u, v) from T^*. Thus, $c(u, v) \leq c(x, y)$ by the cut optimality condition. Hence, $T'' = (T' \setminus (x, y)) \cup (u, v)$ is also a minimum spanning tree, contradicting the assumption on T'. □

The following algorithm is designed based on cut optimality condition.

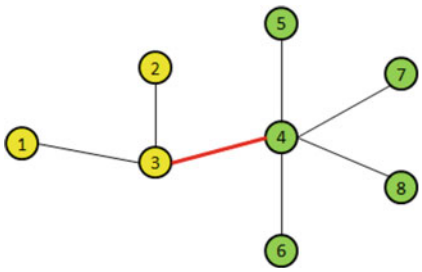

Fig. 4.6 A cut induced by deleting an edge from a spanning tree

Prim Algorithm
input: A graph $G = (V, E)$ with nonnegative edge weight $c :\rightarrow R_+$.
output: A spanning tree T.
  $U \leftarrow \{s\}$ for some $s \in V$;
  $T \leftarrow \emptyset$;
  while $U \neq V$ **do**
    find the minimum weight edge (u, v) from cut $(U, V \setminus U)$
    and $T \leftarrow T \cup (u, v)$;
  return T.

An example for using Prim algorithm is shown in Fig. 4.7. The construction starts at node 1 and guarantees that the cut optimality conditions are satisfied at the end.

The min-priority queue can be used for implementing Prim algorithm to obtain the following result.

Theorem 4.3.2 *Prim algorithm can construct a minimum spanning tree in $O(m \log m)$ time where m is the number of edges in input graph.*

Proof Prim algorithm can be implemented by using min-priority queue in the following way:

- Keep to store all edges in a cut (U, W) in the min-priority queue S.
- At each iteration, choose the minimum weight edge (u, v) in the cut (U, W) by using operation Extract-Min(S) where $u \in U$ and $v \in W$.
- For every edge (x, v) with $x \in U$, delete (c, v) from S. This needs a new operation on min-priority queue, which runs $O(m)$ time.
- Add v to U.
- For every edge (v, y) with $y \in V \setminus U$, insert (v, y) into priority queue. This also requires $O(\log m)$ time.

In this implementation, Prim algorithm runs in $O(m \log m)$ time. □

Prim algorithm can be considered as a local-information greedy algorithm. Actually, its correctness can also be established by an exchange property and a self-reducibility as follows.

Lemma 4.3.3 (Exchange Property) *Consider a cut (U, W) in a graph $G = (V, E)$. Suppose edge e has the smallest weight in cut (U, W). If a minimum spanning tree T does not contain e, then there must exist an edge e' in T such that $(T \setminus e') \cup e$ is still a minimum spanning tree.*

Lemma 4.3.4 (Self-Reducibility) *Suppose T is a minimum spanning tree of a graph G and edge e in T has the smallest weight in the cut induced by deleting e from T. Let G' and T' be obtained from G and T, respectively, by shrinking e into a node. Then T' is a minimum spanning tree of G'.*

We leave proofs of them as exercises.

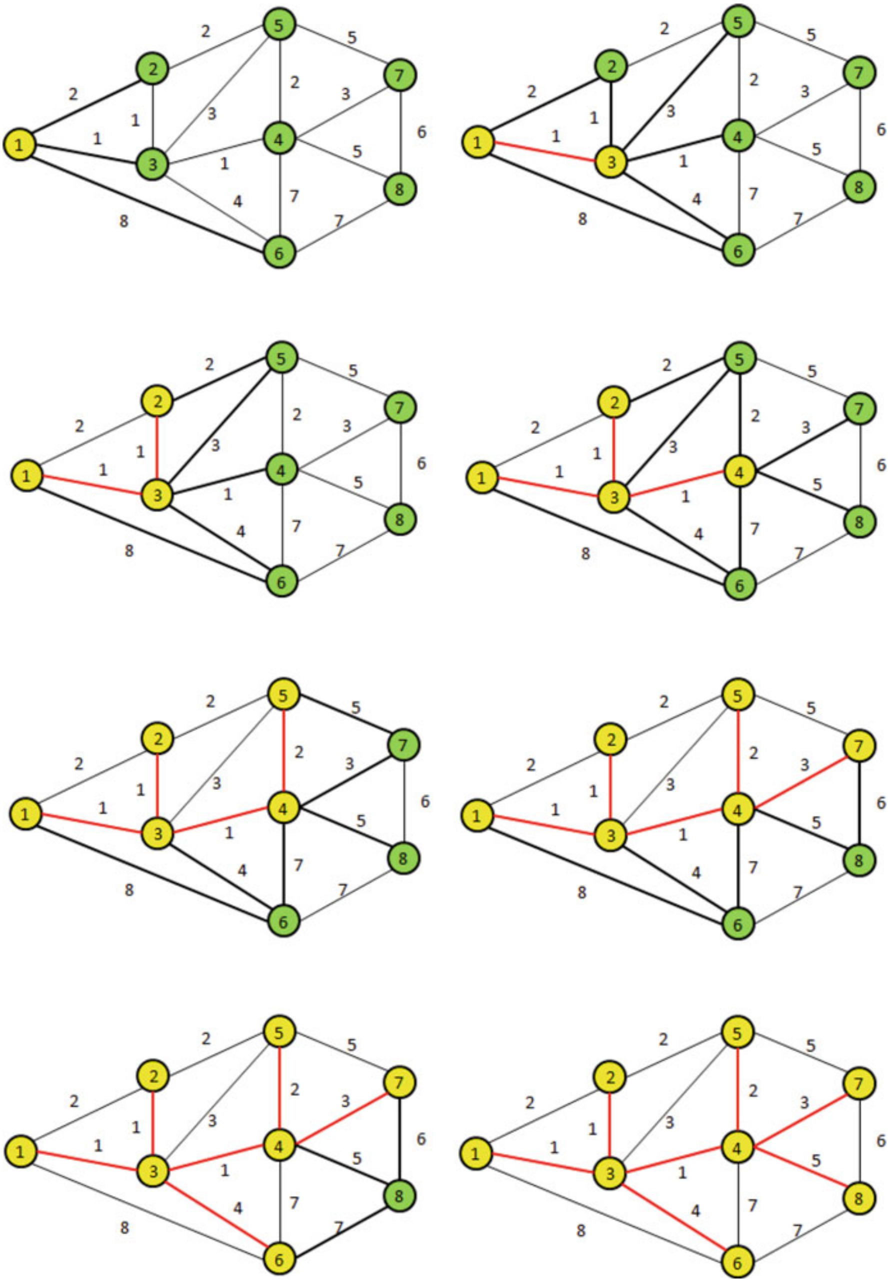

Fig. 4.7 An example with Prim algorithm

4.4 Local Ratio Method

The local ratio method is also a type of algorithm with self-reducibility. Its basic idea is as follows.

Lemma 4.4.1 *Let* $c(x) = c_1(x) + c_2(x)$. *Suppose* x^* *is an optimal solution of* $\min_{x\in\Omega} c_1(x)$ *and* $\min_{x\in Omega} c_2(x)$. *Then* x^* *is an optimal solution of* $\min_{x\in\Omega} c(x)$. *The similar statement holds for the maximization problem.*

Proof For any $x \in \Omega$, $c_1(x) \geq c_1(x^*)$, $c_2(x) \geq c_2(x^*)$, and hence $c(x) \geq c(x^*)$. □

Usually, the objective function $c(x)$ is decomposed into $c_1(x)$ and $c_2(x)$ such that optimal solutions of $\min_{x\in\Omega} c_1(x)$ constitute a big pool so that the problem is reduced to find an optimal solution of $\min_{x\in\Omega} c_2(x)$ in the pool. In this section, we present two examples to explain this idea.

First, we study the following problem.

Problem 4.4.2 (Weighted Activity Selection) Given n activities each with a time period $[s_i, f_i)$ and a positive weight w_i, find a nonoverlapping subset of activities to maximize the total weight.

Suppose, without loss of generality, $f_1 \leq f_2 \leq \cdots \leq f_n$. First, we consider a special case that for every activity $[s_i, f_i)$, if $s_i < f_1$, i.e., activity $[s_i, f_i)$ overlaps with activity $[s_1, f_1)$, then $w_i = w_1 > 0$, and if $s_i \geq f_1$, then $w_i = 0$. In this case, every feasible solution containing an activity overlapping with $[s_1, f_1)$ is an optimal solution. Motivated from this special case, we may decompose the problem into two subproblems. The first one is in the special case, and the second one has weight as follows

$$w_i' = \begin{cases} w_i - w_1 & \text{if } s_i < f_1, \\ w_i & \text{otherwise.} \end{cases}$$

In the second subproblem obtained from the decomposition, some activity may have non-positive weight. Such an activity can be removed from our consideration because putting it in any feasible solution would not increase the total weight. This operation would simplify the problem by removing at least one activity. Repeat the decomposition and simplification until no activity is left.

To explain how to obtain an optimal solution, let A' be the set of remaining activities after the first decomposition and simplification and Opt' is an optimal solution for the weighted activity selection problem on A'. Since simplification does not effect the objective function value of optimal solution, Opt' is an optimal solution of the second subproblem in the decomposition. If Opt' contains an activity overlapping with activity $[s_1, f_1)$, then Opt' is also an optimal solution of the first subproblem, and hence by Lemma 4.4.1, Opt' is an optimal solution for the weighted activity selection problem on original input A. If Opt' does not contain an activity overlapping with $[s_1, f_1)$, then $Opt' \cup \{[s_1, f_1)\}$ is an optimal solution for the first subproblem and the second subproblem and hence also an optimal solution for the original problem.

Based on the above analysis, we may construct the following algorithm.

Local Ratio Algorithm for Weighted Activity Selection
input $A = \{[s_1, f_1), [s_2, f_2), \ldots, [s_n, f_n)\}$ with $f_1 \le f_2 \le \cdots \le f_n$.
$B \leftarrow \emptyset$.
output Opt.
  while $A \neq \emptyset$ **do begin**
    $[s_j, f_j) \leftarrow \text{argmin}_{[s_i, f_i) \in A} f_i$;
    $B \leftarrow B \cup \{[s_j, f_j)\}$;
    for every $[s_i, f_i) \in A$ **do**
      if $s_i < f_j$ **then** $w_i \leftarrow w_i - w_j$;
    end-for
    for every $[s_i, f_i) \in A$ **do**
      if $w_i \le 0$ **then** $A \leftarrow A - \{[s_i, f_i)\}$;
    end-for
  end-while;
  $[s_k, f_k) \leftarrow \text{argmax}_{[s_i, f_i) \in B} f_i$;
  $Opt \leftarrow \{[s_k, f_k)\}$;
  $B \leftarrow B - \{[s_k, f_k)\}$;
  while $B \neq \emptyset$ **do**
    $[s_h, f_h) \leftarrow \text{argmax}_{[s_i, f_i) \in B} f_i$;
    if $s_k \ge f_h$,
      then $Opt \leftarrow Opt \cup \{[s_h, f_h)\}$
        and $[s_k, f_k) \leftarrow [s_h, f_h)$;
    end-if
    $B \leftarrow B - \{[s_h, f_h)\}$;
  end-while;
  return Opt.

Now, we run this algorithm on an example as shown in Fig. 4.8.

Next, we study the second example.

Consider a directed graph $G = (V, E)$. A subgraph T is called an *arborescence* rooted at a vertex r if T satisfies the following two conditions:

(a) If it ignores direction on every arc, then T is a tree.
(b) For any vertex $v \in V$, T contains a directed path from r to v.

Let T be an arborescence with root r. Then for any vertex $v \in V - \{r\}$, there is exactly one arc coming to v. This property is quite important.

Lemma 4.4.3 *Suppose T is obtained by choosing one incoming arc at each vertex $v \in V - \{r\}$. Then T is an arborescence if and only if T does not contain a directed cycle.*

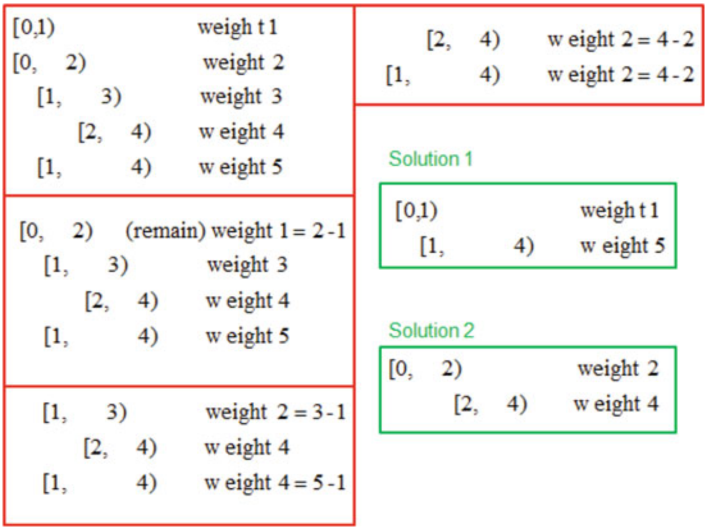

Fig. 4.8 An example for weighted activity selection

Proof Note that the number of arcs in T is equal to $|V| - 1$. Thus, condition (b) implies the connectivity of T when ignore direction, which implies condition (a). Therefore, if T is not an arborescence, then condition (b) does not hold, i.e., there exists $v \in V - \{r\}$ such that there does not exist a directed path from r to v. Now, T contains an arc (v_1, v) coming to v with $v_1 \neq r$, an arc (v_2, v_1) coming to v_1 with $v_2 \neq v$, and so on. Since the directed graph G is finite. The sequence $(v, v_1, v_2, \ldots)$ must contain a cycle.

Conversely, if T contains a cycle, then T is not an arborescence by the definition. This completes the proof of the lemma. □

Now, we consider the minimum arborescence problem.

Problem 4.4.4 (Minimum Arborescence) Given a directed graph $G = (V, E)$ with positive arc weight $w : E \to R^+$ and a vertex $r \in V$, compute an arborescence with root r to minimize total arc weight.

The following special case gives a basic idea for a local ratio method.

Lemma 4.4.5 *Suppose for each vertex $v \in V - \{r\}$ all arcs coming to v have the same weight. Then every arborescence with root r is optimal for the* MIN ARBORESCENCE *problem.*

Proof It follows immediately from the fact that each arborescence contains exactly one arc coming to v for each vertex $v \in V - \{r\}$. □

Since arcs coming to r are useless in construction of an arborescence with root r, we remove them at the beginning. For each $v \in V - \{r\}$, let w_v denote the minimum weight of an arc coming to v. By Lemma 4.4.5, we may decompose the minimum arborescence problem into two subproblems. In the first one, every arc coming to a vertex v has weight w_v. In the second one, every arc e coming to a vertex v has weight $w(e) - w_v$, so that every vertex $v \in V - \{r\}$ has a coming arc with weight 0. If all 0-weight arcs contain an arborescence T, then T must be an optimal solution for the second subproblem and hence also an optimal solution for the original problem. If not, then by Lemma 4.4.3, there exists a directed cycle with weight 0. Contract this cycle into one vertex. Repeat the decomposition and the contraction until an arborescence with weight 0 is found. Then in backward direction, we may find a minimum arborescence for the original weight. An example is shown in Fig. 4.9.

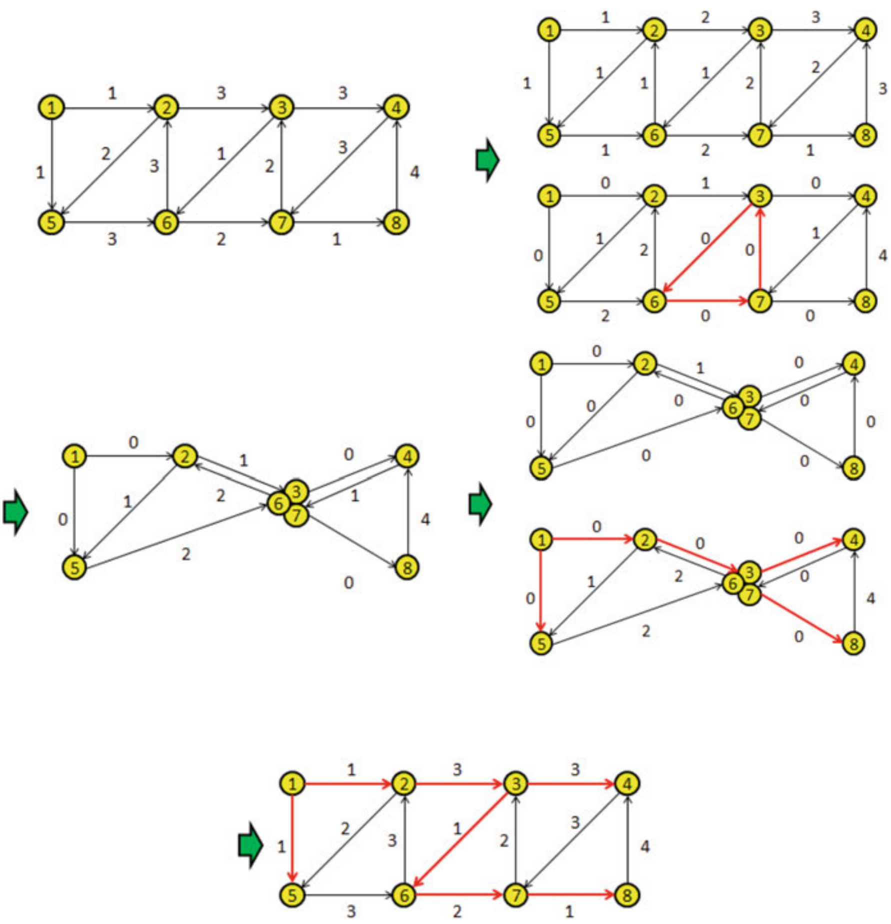

Fig. 4.9 An example for computing a minimum arborescence

According to above analysis, we may construct the following algorithm.

Local Ratio Algorithm for Minimum Arborescence
input a directed graph $G = (V, E)$ with arc weight $w : E \to R^+$,
and a root $r \in V$.
output An arborescence T with root r.
$\mathcal{C} \leftarrow \emptyset$;
repeat
for every $v \in V \setminus \{r\}$ **do**
let e_v be the one with minimum weight among arcs coming
to v and $T \leftarrow T \cup \{e_v\}$;
for every edge $e = (u, v)$ coming to v **do**
$w(e) \leftarrow w(e) - w_v$;
end-for
end-for
if T contains a cycle C
then $\mathcal{C} \leftarrow \mathcal{C} \cup \{C\}$ and
contract cycle C into one vertex in G and T;
end-if
until T does not contain a cycle;
for every $C \in \mathcal{C}$ **do**
add C into T and properly delete an arc of C.
end-for
return T.

Exercises

1. Suppose that for every cut of the graph, there is a unique light edge crossing the cut. Show that the graph has a unique minimum spanning tree. Does the inverse hold? If not, please give a counterexample.
2. Consider a finite set S. Let $\mathcal{I}_k$ be the collection of all subsets of S with size at most k. Show that $(S, \mathcal{I}_k)$ is a matroid.
3. Solve the following instance of the unit-time task scheduling problem.

a_i	1	2	3	4	5	6	7
d_i	4	2	4	3	1	4	6
w_i	70	60	50	40	30	20	10

   Please solve the problem again when each penalty w_i is replaced by $80 - w_i$.
4. Suppose that the characters in an alphabet is ordered so that their frequencies are monotonically decreasing. Prove that there exists an optimal prefix code whose codeword length are monotonically increasing.
5. Show that if $(S, \mathcal{I})$ is a matroid, then $(S, \mathcal{I}')$ is a matroid, where

$$\mathcal{I}' = \{A' \mid S - A' \text{ contains some maximal } A \in \mathcal{I}\}.$$

That is, the maximal independent sets of $(S, \mathcal{I}')$ are just complements of the maximal independent sets of $(S, \mathcal{I})$.

6. Suppose that a set of activities are required to schedule in a large number of lecture halls. We wish to schedule all the activities using as few lecture halls as possible. Give an efficient greedy algorithm to determine which activity should use which lecture hall.
7. Consider a set of n files, $f_1, f_2, \ldots, f_n$, of distinct sizes $m_1, m_2, \ldots, m_n$, respectively. They are required to be recorded sequentially on a single tape, in some order, and retrieve each file exactly once, in the reverse order. The retrieval of a file involves rewinding the tape to the beginning and then scanning the files sequentially until the desired file is reached. The *cost* of retrieving a file is the sum of the sizes of the files scanned plus the size of the file retrieved. (Ignore the cost of rewinding the tape.) The *total cost* of retrieving all the files is the sum of the individual costs.

   (a) Suppose that the files are stored in some order $f_{i_1}, f_{i_2}, \ldots, f_{i_n}$. Derive a formula for the total cost of retrieving the files, as a function of n and the m_{i_k}'s.
   (a) Describe a greedy strategy to order the files on the tape so that the total cost is minimized, and prove that this strategy is indeed optimal.

8. In merge sort, the merge procedure is able to merge two sorted lists of lengths n_1 and n_2, respectively, into one by using $n_1 + n_2$ comparisons. Given m sorted lists, we can select two of them and merge these two lists into one. We can then select two lists from the $m - 1$ sorted lists and merge them into one. Repeating this step, we shall eventually end up with one merged list. Describe a general algorithm for determining an order in which m sorted lists $A_1, A_2, \ldots, A_m$ are to be merged so that the total number of comparisons is minimum. Prove that your algorithm is correct.
9. Let $G = (V, E)$ be a connected undirected graph. The distance between two vertices x and y, denoted by $d(x, y)$, is the number of edges on the shortest path between x and y. The *diameter* of G is the maximum of $d(x, y)$ over all pairs (x, y) in $V \times V$. In the remainder of this problem, assume that G has at least two vertices.

   Consider the following algorithm on G: Initially, choose arbitrarily $x_0 \in V$. Repeatedly, choose x_{i+1} such that $d(x_{i+1}, x_i) = \max_{v \in V} d(v, x_i)$ until $d(x_{i+1}, x_i) = d(x_i, x_{i-1})$.

   Can this algorithm always terminate? When it terminates, is $d(x_{i+1}, x_i)$ guaranteed to equal the diameter of G? (Prove or disprove your answer.)
10. Consider a graph $G = (V, E)$ with positive edge weight $c : E \to R^+$. Show that for any spanning tree T and the minimum spanning tree T^*, there exists a one-to-one onto mapping $\rho : E(T) \to E(T^*)$ such that $c(\rho(e)) \leq c(e)$ for every $e \in E(T)$ where $E(T)$ denotes the edge set of T.

11. Consider a point set P in the Euclidean plane. Let R be a fixed positive number. A steinerized spanning tree on P is a tree obtained from a spanning tree on P by putting some Steiner points on its edges to break them into pieces each of length at most R. Show that the steinerized spanning with minimum number of Steiner points is obtained from the minimum spanning tree.
12. Consider a graph $G = (V, E)$ with edge weight $w : E \to R^+$. Show that the spanning tree T which minimizes $\sum_{e \in E(T)} \|e\|^\alpha$ for any fixed $1 < \alpha$ is the minimum spanning tree, i.e., the one which minimizes $\sum_{e \in E(T)} \|e\|$.
13. Let $\mathcal{B}$ be the family of all maximal independent subsets of an independent system $(E, \mathcal{I})$. Then $(E, \mathcal{I})$ is a matroid if and only if for any nonnegative function $c(\cdot)$, Algorithm 14 produces an optimal solution for the problem $\min\{c(I) \mid I \in \mathcal{B}\}$.
14. Consider a complete bipartite graph $G = (U, V, E)$ with $|U| = |V|$. Let $c(\cdot)$ be a nonnegative function on E such that for any $u, u' \in V_1$ and $v, v' \in V_2$,

$$c(u, v) \geq \max(c(u, v'), c(u', v)) \Longrightarrow c(u, v) + c(u', v') \geq c(u, v') + c(u', v).$$

    (a) Design a greedy algorithm for problem $\max\{c(\cdot) \mid I \in \mathcal{I}\}$.
    (b) Design a greedy algorithm for problem $\min\{c(\cdot) \mid I \in \mathcal{I}\}$.

15. Given n intervals $[s_i, f_i)$ each with weight $w_i \geq 0$, design an algorithm to compute the maximum weight subset of disjoint intervals.
16. Give a counterexample to show that an independent system with all maximal independent sets of the same size may not be a matroid.
17. Consider the following scheduling problem. There are n jobs, $i = 1, 2, \ldots, n$, and there is one super-computer and n identical PCs. Each job needs to be pre-processed first on the supercomputer and then finished by one of the PCs. The time required by job i on the supercomputer is p_i for $i = 1, 2, \ldots, n$; the time required on a PC for job i is f_i for $i = 1, 2, \ldots, n$. Finishing several jobs can be done in parallel since we have as many PCs as there are jobs. But the supercomputer processes only one job at a time. The input to the problem is the vectors $p = [p_1, p_2, \ldots, p_n]$ and $f = [f_1, f_2, \ldots, f_n]$. The objective of the problem is to minimize the completion time of last job (i.e., minimize the maximum completion time of any job). Describe a greedy algorithm that solves the problem in $O(n \log n)$ time. Prove that your algorithm is correct.
18. Consider an independent system $(S, \mathcal{C})$. For a fixed $A \in \mathcal{C}$, define $\mathcal{C}_A = \{B \subseteq S \mid A \setminus B \neq \emptyset\}$. Prove that $(S, \mathcal{C}_A)$ is a matroid.
19. Prove that every independent system is an intersection of several matroids, that is, for every independent system $(S, \mathcal{C})$, there exist matroids $(S, \mathcal{C}_1)$, $(S, \mathcal{C}_2)$, $\ldots (S, \mathcal{C}_k)$ such that $\mathcal{C} = \cap_{i=1}^k \mathcal{C}_i$.
20. Suppose that an independent system $(S, \mathcal{C})$ is the intersection of k matroids. Prove that for any subset $F \subseteq S$, $u(F)/v(F) \leq k$ where $u(F)$ is the cardinality of maximum independent subset of F and $v(F)$ is the minimum cardinality of maximal independent subset of F.

21. Design a local ratio algorithm to compute a minimum spanning tree.
22. Consider a graph $G = (V, E)$ with edge weight $w : E \to Z$ and a minimum spanning tree T of G. Suppose the weight of an edge $e \in T$ is increased by an amount $\delta > 0$. Design an efficient algorithm to find a minimum spanning tree of G after this change.
23. Consider a graph $G = (V, E)$ with distinct edge weights. Suppose that a minimum spanning tree T is already computed by Prim algorithm. A new edge (u, v) (not in E) is being added to the graph. Please write an efficient algorithm to update the minimum spanning tree. Note that no credit is given for just computing a minimum spanning tree for graph $G' = (V, E \cup \{(u, v)\})$.
24. Consider a matroid $\mathcal{M} = (X, \mathcal{I})$. Each minimal dependent set C is called a *circuit*. A *cut* D is a minimal set such that D intersects every base. Suppose that a circuit C intersects a cut D. Show that $|C \cap D| \geq 2$.

Historical Notes

The greedy algorithm is an important class of computer algorithms with self-reducibility, for solving combinatorial optimization problems. It uses the greedy strategy in construction of an optimal solution. There are several variations of greedy algorithms, e.g., Prim algorithm for minimum spanning tree in which greedy principal applies not globally but a subset of edges.

Could Prim algorithm be considered as a local search method? The answer is no. Actually, in a local search method, a solution is improved by finding a better one within a local area. Therefore, the greedy strategy applies to search for the best moving from a solution to another better solution. This can also be called as incremental method, which will be introduced in the next chapter.

The minimum spanning tree has been studied since 1926 [30]. Its history can be found a remarkable article [185]. The best known theoretical algorithm is due to Bernard Chazelle [49, 50]. The algorithm runs almost in $O(m)$ time. However, it is too complicated to implement and hence may not be practical.

Matroid was first introduced by Hassler Whitney in 1935 [406] and independently by Takeo Nakasawa [329]. It is an important combinatorial structure to describe the independence with axioms. Especially, those axioms provide an abstraction for common properties in linear algebra and graphs. Therefore, many concepts and terminologies are analogous in these two areas. The relationship between matroid and greedy algorithm is only a small portion in the theory of matroid [334, 384, 403]. Actually, the study of a matroid contains a much larger field, with connections to many topics [404], such as combinatorial geometry [37, 74, 405], unimodular matrices [171], projective geometry [308], electrical networks [316, 348], and software systems [254].