Backtracking is a generic method that can be applied to many problems. Finding a backtracking algorithm can often be a first step towards finding a greedy or dynamic-programming algorithm.

However, backtracking does not usually result in optimal algorithms or dramatic running-time improvements over the naive approach. Furthermore, exact time analysis can be difficult, since it is difficult to exactly determine the range of the search.

Examples of backtracking include depth-first search, branch-and-bound, and other similar approaches that perform a search by exhaustive case analysis.

**Example: Maximal Independent Set**

Given a graph with nodes representing people, and edges representing enmities, find the largest set of people having no mutual animosity, i.e. the largest set of unconnected nodes.

**Greedy approach**

While some people remain, choose the person with the fewest enemies.

```plaintext
1   S <- { all nodes of G }
2   P <- { }   
3   while S not empty
4       s <- node in S with smallest degree
5       add(P, s)
6       S <- S - s - adj(s)
```

This is fast. However, it doesn’t always find the best solution, because it is possible to back oneself into an algorithmic corner by making a choice that prevents you from making better choices later.

**Exhaustive search**

Make up a list of every possible subset of people, and choose the largest compatible subset. This will find the solution, but requires exponential time ($O(2^n)$).
Backtracking approach

This exhaustive approach is wasteful, though: by choosing one person, we eliminate all of its neighbors at once – i.e. $2^{\lvert \text{adj}(s) \rvert}$ rows of the exhaustive truth table can be handled equivalently. The backtracking approach systematizes this process of exhaustive, sequential choice as a tree of decisions.

More precisely, let $G = (V, E)$ be an undirected graph. We want to find a set $I \subseteq V$ such that if $(u, v) \in E$, then either $u \notin I$ or $v \notin I$. Call this $I$ a maximal independent set for $G$ (note that $I$ may not be unique). The backtracking approach finds $I$ with the following algorithm:

1. $\text{MIS}(G = (V, E))$
2. if $V = \text{NIL}$
   3. return $\text{NIL}$
3. if $|V| = 1$
   4. return $V$
5. for $x$ in $V$
6.     $S_{in} \leftarrow \text{union}(\text{MIS}(G - x - \text{adj}(x)), x)$
7.     $S_{out} \leftarrow \text{MIS}(G - x)$
8. return largest($S_{in}$, $S_{out}$)

This takes $T(n) \leq 2T(n-1) + O(1)$, since each of the two recursive calls has size at most $n - 1$. We can’t apply the master theorem (why?), but by noting that in the worst-case the call tree forms a complete binary tree of depth $n$, $\text{MIS}$ is $O(2^n)$.

While this is not better than the exhaustive approach above, a more careful implementation can yield a win by handling a special case: if $x$ has degree 0, then it should always be included in $I$, so we can skip the recursive call on line 8. So the recurrence becomes $T(n) \leq T(n-1) + T(n-2) + O(1)$ (hm.. Fibonacci strikes again), i.e. $T(n) = O(F_n) = O\left((1+\sqrt{5})/2\right)^n \approx O(2^{0.7n})$. We can prove the magic number by guess-and-check, guessing that $T(n)$ is exponential in $n$:

$$
T(n) = \omega^n
= T(n-1) + T(n-2) = \omega^{n-1} + \omega^{n-2}
\implies \omega^2 - \omega - 1 = 0
\implies \omega = \frac{1 + \sqrt{5}}{2}
$$

Aside: DES Challenge

To get some idea how much of an improvement $2^{0.7n}$ is over $2^n$, the DES challenge cracked a 56-bit key by exhaustive search over $2^{56}$ using 1000 computers and 3 months. At this rate, breaking an 80-bit key takes $2^{80}$ steps ($56/0.7$), or 16 million times more resources. You might as well just set your computers on fire. In the other direction, $2^{40}$ ($56 \times 0.7$) would take only a few hours.
In other words, those constants in the exponent matter! Can we find other heuristics to improve the constant? The current worst-case graph is a straight line of linked nodes. In this case, we really only need to consider either the even or the odd elements in the “list”.

Generalizing this idea, if \( x \) has only a single neighbor \( y \) in \( G \), then we can return the MIS for \( G - \{x\} \), which must be of the same size as \( \text{union}(\text{MIS}(G - x - y), x) \). This reduces the running time to \( O(2^{0.6n}) \). However, we don’t have an instance of a graph exhibiting this worst-case behavior, so we might be able to do better.