Recursive algorithm for Maximum Independent Set Problem

We will see an example of an algorithm that can be analyzed by “guess and verify” method similar to Fibonacci recurrence.

The input to our problem is an undirected graph \((V, E)\), where \(V\) is a collection of nodes (we can assume \(V = \{0, 1, \ldots, n - 1\}\) and \(E\) is a collection of edges, which are unordered pairs of nodes.

If \(A \subseteq V\), we say that \(A\) is independent if it does not contain any edge. Our goal is to find an independent set of the maximum size.

Brute force approach would check all subsets of \(V\), and we have \(2^n\) such subsets. If we have time to perform \(2^{50}\) check, we can handle inputs with 50 nodes.

Algorithms based on case analysis can do better, the best runs in time \(2^{n/4}\), and it uses thousands of computer verified cases. An algorithm with seven cases runs in time \(O(2^{0.41n})\).

Notation: \(N(u, A)\) is the set of neighbors of \(u\) in \(A\), i.e. the nodes in \(A\) connected to \(u\) with edges
Our recursive function $MIS(A)$ returns a maximum independent set contained in $A$, where $A$ is a subset of $V$.

The first two cases reduce the problem to a single smaller instance.

Case 1: for some node $u \in A$, $B = N(u, A)$ is a clique, i.e. every two nodes in $B$ are connected with an edge.

Comment: if a maximum independent set $A$ does not contain $u$, we also have a maximum independent set $\{u\} \cup A - B$ that does

```plaintext
return $\{u\} \cup MIS(A - \{u\} - B)$
```

Case 2: for some node $u \in A$, $B = N(u, A)$ has two nodes and $C = N(N(u, A), A)$ also has two nodes.

Comment: because Case 2 does not apply, $B$ has exactly two nodes, and these nodes are not connected with an edge; if a maximum independent set $A$ does not contain $B$, $B \cup A - C$ is a maximum independent set that does

```plaintext
return $B \cup MIS(A - B - C)$
```
Case 3: for some node \( u \in A \), \( N(u, A) \) has two nodes, define \( B, C \) as in Case 2

Comment: if a maximum independent set \( A \) does not contain \( u \) and it contains only one node in \( B \), we can have \( \{u\} \cup A - B \) instead

return the larger of those two:

\[
\{u\} \cup MIS(A - \{u\} - B), \quad \text{and} \quad B \cup MIS(A - B - C)
\]

Let \( T(n) \) be the running time of \( MIS(A) \) as a function of the number of nodes in \( A \).

We can show this case is consistent with \( T(n) \leq 2^{0.41n} \): because Case 2 does not apply, \( C \) has at least 3 nodes, so the first recursive call has 3 nodes fewer than \( A \), and the second, 5 fewer nodes, and this leads to

\[
T(n) \leq T(n - 3) + T(n - 5) \leq 2^{0.41(n-3)} + 2^{0.41(n-5)} = 2^{0.41n} (2^{-1.23n} + 2^{-2.05n}) < 2^{0.41n} (0.427 + 0.242)
\]
Case 4: for some node $u \in A$, \( B = N(u, A) \) has at least four nodes
return the larger of those two:
\[ MIS(A - \{u\}), \] and
\[ \{u\} \cup MIS(A - \{u\} - B) \]

Again, we show this case is consistent with \( T(n) \leq 2^{0.41n} \): the first recursive call has 1 fewer node, and the second, 5 fewer nodes:

\[
T(n) \leq T(n - 1) + T(n - 5) \leq \\
2^{0.41(n-1)} + 2^{0.41(n-5)} = \\
2^{0.41n}(2^{-0.41n} + 2^{-2.05n}) < \\
2^{0.41n}(0.753 + 0.242)
\]

In the remaining cases, each node has exactly 3 neighbors.
Case 5: for some node $u \in A$, $B = N(u, A)$ contains an edge, $v \in B$ is not in this edge, $C = N(v, A)$

Comment: if a maximum independent set $A$ does not contain $u$ and it contains only one node in $B$, we could replace it with a set that does contain $B$; in the remaining case $A$ contains two nodes in $B$, hence it contains $v$

return the larger of those two:

\[
\begin{align*}
\{u\} & \cup MIS(A - \{u\} - B) \\
\{v\} & \cup MIS(A - \{v\} - C)
\end{align*}
\]

Both recursive calls have 4 fewer nodes.
Case 6: for some node $u, v \in A$, $B = N(u, A)$ and $C = N(v, A)$ have at least two common nodes.

Comment: because case 5 does not apply, $u$ and $v$ are not connected; if a maximum independent set contain $u$ but it does not contain $v$, we can insert $v$ and remove $C$, this removes only $C - B$, at most one node.

return the larger of those two:

$MIS(A - \{u, v\})$, and

$\{u, v\} \cup MIS(A - \{u, v\} - B - C)$

One recursive call has 2 fewer nodes, and the other, at least 5 fewer nodes.
Case 7: the remaining case
Comment: every $u \in A$ is connected to three other nodes $v, w, x$ which share no other neighbors

return the largest of those four:

- $\{u\} \cup \text{MIS}(A - \{u\} - N(u, A))$
- $\{v, w\} \cup \text{MIS}(A - \{v, w\} - N(v, A) - N(w, A))$
- $\{v, x\} \cup \text{MIS}(A - \{v, x\} - N(v, A) - N(x, A))$
- $\{w, x\} \cup \text{MIS}(A - \{w, x\} - N(w, A) - N(x, A))$

The first recursive call has 4 fewer nodes, and the other three, 7 fewer nodes, and this is also consistent with $T(n) \leq 2^{0.41n}$:

$$T(n) \leq T(n - 4) + 3T(n - 7) \leq 2^{0.41(n-4)} + 3 \times 2^{0.41(n-7)} = 2^{0.41n} (2^{-1.64n} + 3 \times 2^{-2.87n}) < 2^{0.41n}(0.321 + 3 \times 0.137)$$

Only the case when we eliminate a node with 4 neighbors is tight, this makes this algorithm very easy to improve.
Examples of brute force algorithms

It makes sense to look at “brute force” algorithms for several reasons.

• It is easy (easier) to write a correct brute force algorithm. When we want to show that a “clever” algorithm is correct, an approach is to show that it is equivalent to a brute force algorithm.

• We can be a bit clever: stop working if we see a chance. Sometime it can be a good algorithm in practise (average case etc.), sometimes, “good enough”. After all, programming time may be a bottleneck resource.

• To appreciate clever algorithms, we need to know the alternative.
Example 1: insertion sort

Statement of sorting problem:

\((A, a, b)\) consists of \(A[a], A[a+1], \ldots, A[b-1]\)

rearrange entries in \((A, a, b)\) so \(A[a] \leq A[a+1] \leq \ldots \leq A[b-1]\).

for \((i = b-1; \ i > a; \ i--)\) {
    \(p = A[i-1];\)
    for \((j = i; \ j < b; \ j++)\)
        if \((A[j] < p)\)
        else
            break;
    \(A[j-1] = p;\)
}

Does it work? Idea:

consider a “run” of the outer loop with a particular value of \(i\)
before that run, the entries \(A[i], \ldots, A[b-1]\) are already sorted
now we want to put \(p = A[i-1]\) correctly in that order
we shift the initial part, those that are smaller than \(p\), one position to the “left”
when we are done with the shifting, either because of \(break\) or because we reached \(j == b\), \(j-1\) is the correct place for \(p\)
Example 2: String Matching

Statement of the string matching problem:
we have pattern \((P, 0, m)\) and text \((T, 0, n)\).
Report every \(i\) such that \(P[j] == T[i+j]\) for \(j = 0, \ldots, m-1\)
\[
\text{for } (i == 0; \; i <= n-m; \; i++) \{
\text{for } (j == 0; \; j < m; \; j++)
\text{if } (P[j] != T[j+i])
\text{break;}
\text{if } (j == m)
\text{report } i;
\}
\]
For a random pattern, this is actually efficient. If both the pattern and the text
have all entries identical, we perform \(\Omega(mn)\) steps.
Example 3: unmasking extremists

Each member of Congress $i$ has a personal freedom rating $x_i$ and economic freedom rating $y_i$. We can say the member $i$ is an extremist if for some coefficients $a, b$ the value $ax_i + bx_i$ is larger than for any other member.

How can we identify extremists? It is a special, two-dimensional case of a more general problem of finding extremal points. Clearly, trying every possible pair $(a, b)$ would take forever (or longer, if you allow uncountably many values).

So the question is: can we analyze a single member (point) if some finite time? Let us first shift the coordinate system to place that point at the origin. This is OK, because for each linear function $ax + by$, the shift adds the same constant to all values, so we have the same extremists as before. However, our point gets value 0 for EVERY pair $(a, b)$, so:

this is an extremal point if for every $(a, b)$ there is a point $(x, y)$ with positive $ax + by$.

Easy case: there are points in each of four quadrants. Our point is not extremal.

Easy case: all points are in one quadrant, or in two adjacent quadrants.

A case: we have points in quadrants:

I (++), II (+-), III (--) and none in IV (-+).
If a linear function makes our point positive and the rest negative then after adding a constant it makes our point zero and the rest negative. So it is $ax + by$.

If $a, b \geq 0$, all points in I quadrant are positive
not a good choice
If $a, b \leq 0$, all points in III quadrant are positive
not a good choice
If $a \leq 0$, $b \geq 0$, all points in II quadrant are positive.
not a good choice
So we assume that $a > 0$, $b < 0$, and the sign of $ax + by$ is the same as the sign of $x - \beta y$ for some $\beta > 0$

Thus we test if there exists some $\beta$ that would make all points in quadrants I, II and III negative, meaning

$$x - \beta y < 0$$

This is always true in quadrant II

In quadrant I this is equivalent to $x/y < \beta$, so $\beta$ is above the maximum of $x/y$, say $M_I$

In quadrant III this is equivalent to $x/y > \beta$, so $\beta$ is below the minimum of $x/y$, say, $m_{III}$

Such $\beta$ exists if and only if $M_I < m_{III}$, linear time check.
Divide and conquer: merge sort