Backtracking

Lecture12
Recursion

• We have seen divide and conquer:
  — split into subproblems of size \( n/c \) (some \( c \)).
  — Analyze running time with recursion trees.

• Different style of recursion: Backtracking
  — reduce to subproblems of smaller size \( n-c \) (some \( c \)).
  — Usually exponential time
  — Way of developing correct recursive algorithms, won’t deal with running time often.
8-Queens Puzzle
8-Queens Puzzle

How long does it take to solve it from scratch?
n-Queens Puzzle

Represent by array Q[1…n].
Q[i] = which square in row i has a queen
n-Queens Puzzle

Place a queen at the first empty row-try all possible places
n-Queens Puzzle

Place a queen at the first empty row-try all possible places
n-Queens Puzzle

Place a queen at the first empty row-try all possible places
n-Queens Puzzle

Place a queen at the first empty row—try all possible places
n-Queens Puzzle

\textbf{RecursiveNQueens}(Q[1..n], r):

\begin{itemize}
  \item if \( r = n + 1 \)
    \begin{itemize}
    \item print \( Q \)
    \end{itemize}
  \item else
    \begin{itemize}
    \item for \( j \leftarrow 1 \) to \( n \)
      \begin{itemize}
      \item legal \leftarrow \text{TRUE}
      \item for \( i \leftarrow 1 \) to \( r - 1 \)
        \begin{itemize}
        \item if \( (Q[i] = j) \) or \( (Q[i] = j + r - i) \) or \( (Q[i] = j - r + i) \)
          \begin{itemize}
          \item legal \leftarrow \text{FALSE}
          \end{itemize}
        \end{itemize}
      \end{itemize}
      if legal
      \begin{itemize}
      \item \( Q[r] \leftarrow j \)
      \end{itemize}
      \textbf{RecursiveNQueens}(Q[1..n], r + 1)
    \end{itemize}
\end{itemize}
\end{itemize}
n-Queens Puzzle
Subset sum

• Given a set $X$ of positive integers and a target positive integer $t$, is there a subset of elements in $X$ that add up to $t$?

• Given $X$, find a subset of $X$, so that $\sum A = t$?

• What is the first element to go into $A$?

• Try them all!

• If there is an element equal to $t$, done

• If $t$ is zero, we are done! (why?)

• If $t$ is negative, no!
Subset sum

- Given a set $X$ of positive integers and a target positive integer $t$, is there a subset of elements in $X$ that add up to $t$?

- Given $X$, find a subset of $X$, so that $\sum A = t$?

- Assume $t$ is positive and no element bigger than $t$. 
Subset sum

• Given a set $X$ of positive integers and a target positive integer $t$, is there a subset of elements in $X$ that add up to $t$?

• Given $X$, find a subset of $X$, so that $\sum A = t$?

• Example: $X = \{3, 2, 4, 6, 9\}$, $t = 7$

• What element to try first?

• Say $x = 6$. Then is there subset of $\{3, 2, 4, 9\}$ that adds to 1? NO
Subset sum

• Given a set X of positive integers and a target positive integer t, is there a subset of elements in X that add up to t?

• Given X, find a subset of X, so that \( \sum A = t \)?

• Example: \( X = \{3,2,4,6,9\}, \ t = 7 \)

• What element to try first?

• Say \( x = 6 \). Then is there a subset of \( \{3,2,4,9\} \) that adds to 1? NO

• Two cases: \( x \) in A or \( x \) not in A.
Subset sum

• If there is a subset A with $\sum A = t$ then either

• $x$ in A, call SubsetSum($X$-$\{x\}$, $t-x$)

• or $x$ not in A call SubsetSum($X$-$\{x\}$, $t$)
Subset sum

Call the algorithm with i=n
Canonical order to choose elements in the subset
Subset sum

- Running time?

- \( T(n) \leq O(1) + 2T(n-1) \)

- Tower of Hanoi! exponential time \( 2^n \)

- Brute force!

- NP-Hard!
NFA acceptance

• Given NFA: \( N = (\Sigma, Q, \delta, s, A) \) and \( w \in \Sigma^* \)

\[ \delta^*(s, w) \cap A \neq \emptyset \]

• Is there a walk in \( N \) from \( s \) to an accepting state labeled \( w \)?
NFA acceptance

- Input = 01001

- \( L = \{\text{contains either 00 or 11}\} \)
NFA

\begin{align*}
[s] & \xrightarrow{1} [b] & \xrightarrow{0,1} [s] \\
[a] & \xrightarrow{1} [t] & \xrightarrow{0,1} [a] \\
[b] & \xrightarrow{0} [a] & \xrightarrow{1} [b] \\
\end{align*}

\begin{align*}
[s] & \rightarrow [s] \\
[b] & \rightarrow [b] \\
[a] & \rightarrow [a] \\
[t] & \rightarrow [t] \\
\end{align*}

\begin{align*}
1001 & \rightarrow 1001 & 1001 & 1001 & 1001 & 1001
\end{align*}
One of the states are accepting. There needs to be AT LEAST one accepting state.
NFA acceptance

- Input = 01001

- How do I decide what to do once I read the first 0?

- Try both! maybe one of them will work.

- Smaller subproblem, when we need to figure out if the NFA accepts a smaller input.

- Need to specify what state the NFA is in and what string is left to read.

- Accept (q,w)
NFA acceptance

\[\text{Accepts?}(q, w[1..n]):\]
\[
\begin{align*}
\text{if } n = 0 & \quad \text{return } A[q] \\
\text{for all states } r & \quad \text{if } \delta[q, w[1], r] \text{ and Accepts?}(r, w[2..n]) \\
& \quad \text{return } \text{TRUE}
\end{align*}
\]
\text{return } \text{FALSE}

- \(A[i]\) is 1 iff \(i\) is an accepting state.
- \(\delta[q,w[1],r] = 1\) iff \(r \in \delta(q,w[1])\)
- Every time the recursion branches, there are at most \(Q\) states
- \(Q^n\) upper bound on running time!!!
Longest Increasing Subsequence (LIS)

- 3 1 4 1 5 9 2 6 5 3 8 2 7 9 4 6 1 0 4 8

- Subsequence different than substring.
- Increasing = in an order.
- Recursion?
Longest Increasing Subsequence (LIS)

- 3 1 4 1 5 9 2 6 5 3 8 2 7 9 4 6 1 0 4 8

- Look at first element. Keep or ditch?

- LIS(A[1…n])

  If n < 10^{10}, brute force

  keep: 1 + LIS(A[2…n])
  
ditch: LIS(A[2…n])

What went wrong? I didn’t use INCREASING
Longest Increasing Subsequence (LIS)

- 31415926538279461048

- LIS(A[1…n])

  If n < 10^{10}, brute force
  keep: 1+  ?
  ditch: LIS(A[2…n])

  What is the correct subproblem?
  - LIS where every number is larger than the number p I keep
  - Not the same problem anymore!
Longest Increasing Subsequence (LIS)

- 31415926538279461048

- LIS(A[1...n], p)

If n < 10^{10}, brute force

keep:

- What are the new cases?
- Anything else?

ditch:
Longest Increasing Subsequence (LIS)

- 31415926538279461048

- LIS(A[1…n],p)

  If n< 10^{10}, brute force

  If A[1] ≤ p,

  RETURN LIS(A[2…n],p)

  else

  \[
  \text{RETURN MAX: } 1 + \text{LIS(A[2…n],A[1])}
  \]
Longest Increasing Subsequence (LIS)

- 3 1 4 1 5 9 2 6 5 3 8 2 7 9 4 6 1 0 4 8

- LIS(A[1…n],p)

  If n < 10^{10}, brute force
  If A[1] ≤ p,

  RETURN LIS(A[2…n],p)

  else

  RETURN MAX: 1 + LIS(A[2…n],A[1])

  - LIS(A[1…n],−∞) to find LIS
  - Running time?
  - 2^n