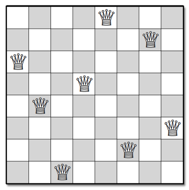
Queens Problem

Specification (search version)

Input: $n \in \mathbb{N}$

Output: Position of n queens on $n \times n$ board such that no two queens threaten each other, i.e., no two queens are on the same row, column, or diagonal.



Model

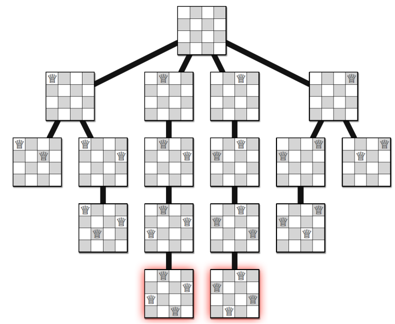
- Components $\sim$ queens. State space: $[n] \times [n]$
- Components $\sim$ rows. State space: $[n]$.

Backtracking

Approach

- Check for violations of constraints by partial solutions.
- Backtrack if violation detected.

Queens problem



Generating All Solutions – backtracking

Extended specification

Input: instance I with solutions of length n
$\ell \in \mathbb{N}$ with $\ell \leq n$
$S[1 \ldots \ell]$ satisfying all constraints involving first ℓ components only

Output: all valid solutions for I that start with $S[1 \ldots \ell]$

```
procedure GenAllExtension(I, n, ℓ, S[1...ℓ])
  if ℓ = n then output S[1...n]
  else
    for each possible setting v do
      S[ℓ+1] ← v
      if setting of S[ℓ+1] induces no constraint violations then
        GenAllExtension(I, n, ℓ+1, S[1...ℓ+1])
```

Common Problem Type

Setting

- System consisting of n components.
- Each component can be in a finite number of states.
- Certain constraints defining which combinations of states are valid.
- Objective function f from settings to $\mathbb{R}$

Goal

Decision: Decide whether a solution exists.

Search: Find a solution.

Generation: Output all solutions.

Count: Output the number of solutions.

Optimal solution: Output solution that maximizes or minimizes f.

Optimal value: Output max or min value of f over valid solutions.

Interval Scheduling

Problem specification (value version)

Input: intervals $I_i = [s_i, f_i)$ and values $v_i \in \mathbb{R}$ for $i \in [n]$.

Output: maximum of $\sum_{i \in S} v_i$ over all $S \subseteq [n]$ such that no intervals I_i and I_j for distinct $i, j \in S$ overlap

Subproblems

Input: $J \subseteq [n]$

Output: $\mathrm{OPT}(J) \doteq$ maximum of $\sum_{i \in S} v_i$ over all $S \subseteq J$ such that no intervals I_i and I_j for distinct $i, j \in S$ overlap

```
procedure MaxVal(J)
  if J = ∅ then return 0
  else
    j* ← min(J)
    C ← {j ∈ J : I_j ∩ I_{j*} ≠ ∅}
    return max(MaxVal(J \ {j*}), v_{j*} + MaxVal(J \ C))
```

Analysis

Correctness

Running time

- Aggregate local work over all subproblems
- Work per subproblem: $O(n)$
- Number of subproblems
  - Number of subsets of $[n]$ equals 2^n.
  - Example where number of distinct subproblems is at least $2^{n/2}$.
  - Can be improved by considering intervals in appropriate order.

Improved Algorithm

Idea

Sort the intervals $I_i = [s_i, f_i)$ by smallest s_i first, then run prior algorithm.

Subproblems

Suffixes of $[n]$, i.e., subsets of the form $\{k, k+1, \dots, n\}$ where $k \in [n+1] \doteq \{1, 2, \dots, n+1\}$

Recurrence

- $\text{OPT}(k) \doteq$ maximum total value achievable by intervals I_j with $j \in \{k, k+1, \dots, n\}$.
- $\text{OPT}(k) = \max(\text{OPT}(k+1), v_k + \text{OPT}(\text{next}(k)))$ where $\text{next}(k) \doteq \min(\{j \in \{k+1, \dots, n\} : s_j \geq f_k\} \cup \{n+1\})$
- Base case: $\text{OPT}(n+1) = 0$
- Answer: $\text{OPT}(1)$

Analysis

Correctness

Running time

- Sorting: $O(n \log n)$
- Number of subproblems: $n+1$
- Amount of work per subproblem: $O(\log n)$ for finding $\text{next}(k)$ using binary search.
- Total: $O(n \log n)$

Memory space

- $O(n)$

Retrieving the Solution

Recursively

Return both the value and a solution achieving it.

Iteratively

```
procedure Retrieve-Solution
    S ← ∅
    k ← 1
    while k ≤ n do
        if OPT(k) = OPT(k + 1) then
            k ← k + 1
        else
            S ← S ∪ {k}
            k ← next(k)
    return S
```

Paradigm

Dynamic Programming

Recursive approach such that:

1. The number of distinct subproblems in the recursion tree remains small.
2. Each of those subproblems is solved only once.

Realizing property 2

- Memoization
- Iteration

Analyzing property 1

- Looking back: What information about backtracking history suffices to continue the process? [state reduction]
- Looking forward: What set of parameters suffice to describe all subproblems? [explicit description of subproblems]

Knapsack Problem

Problem

Input: items $i \in [n]$ specified by weight $w_i \in \mathbb{Z}^+$ and value $v_i \in \mathbb{R}$; weight limit $W \in \mathbb{Z}^+$

Ouput: $S \subseteq [n]$ such that $\sum_{i \in S} w_i \leq W$ and $\sum_{i \in S} v_i$ is maximized.

Principle of optimality

- Case $i^* \notin S$:
  Remains to solve given instance with i^* removed.
- Case $i^* \in S$ (only an option if $w_{i^*} \leq W$):
  Remains to solve given instance with i^* removed and weight limit $W - w_{i^*}$.
- Informally, $\text{OPT}(I)$ is the maximum of:
  - $\text{OPT}(I \text{ without } i^*)$
  - $v_{i^*} + \text{OPT}(I \text{ without } i^* \text{ and weight limit } W - w_{i^*})$.

Dynamic Program for Optimal Value

- Consider items in given order.
- State reduction: $\Theta(n \cdot W)$ states
  last item considered, total weight thus far
- Subproblem specification:
  $\text{OPT}(k, w) = \text{OPT}(\text{items } \{k, \dots, n\} \text{ and weight limit } w)$
  where $1 \leq k \leq n+1$ and $0 \leq w \leq W$
- Recurrence: $\text{OPT}(k, w) =$
  $\max(\text{OPT}(k+1, w), v_k + \text{OPT}(k+1, w - w_k) \text{ only if } w_k \leq w)$
- Base cases ($k = n+1$): $\text{OPT}(n+1, w) = 0$
- Answer: $\text{OPT}(1, W)$
- Evaluation order for iterative implementation

Retrieving the Solution

Pseudocode

procedure RETRIEVE-SOLUTION
  $S \leftarrow \emptyset$
  $w \leftarrow W$
  for $k = 1$ to n **do**
    if $w_k \leq w$ **cand** $\text{OPT}(k, w) = v_k + \text{OPT}(k+1, w - w_k)$
      then $S \leftarrow S \cup \{k\}$; $w \leftarrow w - w_k$
  return S

Complexity analysis

- Time: $O(n \cdot W)$ with or without retrieval.
- Space: $O(n \cdot W)$ with retrieval; $O(W)$ without.

Problem Specifications

Sequence alignment

Input: strings $A[1, \ldots, n]$ and $B[1, \ldots, m]$

Ouput: alignment of A and B that maximizes the number of matches

Longest common subsequence

Input: strings $A[1, \ldots, n]$ and $B[1, \ldots, m]$

Ouput: subsequence of both A and B of maximum length

Principle of optimality

Consider alignment at the end.

- Case 1: Do not align $A[n]$
  Contribution: 0
  Remains to solve problem for $A[1, \ldots, n-1]$ and $B[1, \ldots, m]$
- Case 2: Do not align $B[m]$
  Contribution: 0
  Remains to solve problem for $A[1, \ldots, n]$ and $B[1, \ldots, m-1]$
- Case 3: Align $A[n]$ and $B[m]$
  Contribution: 1 if $A[n] = B[m]$, 0 otherwise
  Remains to solve problem for $A[1, \ldots, n-1]$ and $B[1, \ldots, m-1]$

DP Approach

Subproblems

$\text{OPT}(i, j)$ = length of a longest common subsequence of $A[1, \ldots, i]$ and $B[1, \ldots, j]$ ($0 \leq i \leq n$ and $0 \leq j \leq m$)

Recursion

$\text{OPT}(i, j) =$
$\max(\text{OPT}(i-1, j), \text{OPT}(i, j-1), \delta_{A[i],B[j]} + \text{OPT}(i-1, j-1))$

where $\delta_{a,b} \doteq \begin{cases} 1 & \text{if } a = b \\ 0 & \text{otherwise} \end{cases}$

Base cases

$\text{OPT}(0, j) = 0 = \text{OPT}(i, 0)$

Answer: $\text{OPT}(n, m)$

Interpretation

Finding a longest path from $(0, 0)$ to (n, m) in grid digraph

Complexity Analysis

Time

- $O(nm)$ table entries
- $O(1)$ time per entry
- $O(nm)$ total time

Space

- $O(\min(n, m))$ for length of longest common subsequence
- $O(nm)$ for alignment / longest common subsequence

Reducing Space Complexity for Alignment / LCS

- Need to find longest simple path from $(0, 0)$ to (n, m).
- Path must be in column $m/2$ at least once, say in row i^*.
- Once we know i^*, remains to find:
  (a) longest simple path from $(0, 0)$ to $(i^*, m/2)$, and
  (b) longest simple path from $(i^*, m/2)$ to (n, m).
- Both (a) and (b) are significantly smaller instances of the same problem.
- To find i^* compute for each $i \in [n]$:
  (a) $f(i)$: length of longest simple path from $(0, 0)$ to $(i, m/2)$
  (b) $g(i)$: length of longest simple path from $(i, m/2)$ to (n, m)
  Then set i^* to an $i \in [n]$ that maximizes $f(i) + g(i)$.
- As $f(i) = \text{OPT}(i, m/2)$, all of f can be computed in time $O(nm)$ and space $O(n)$ using original algorithm.
- Same applies to g by symmetry (reverse direction of edges).
- Thus, i^* can be computed in time $O(nm)$ and space $O(n)$.

Complexity Analysis

Space

- $O(n+m)$ for path [global]
- $O(n)$ for computing i^* [local, reused]
- $O(1)$ per level of recursion [recursion stack]
- Total: $O(n+m) + O(n) + O(\log m) = O(n+m)$

Time

- local work: $c \cdot n \cdot m$
- dimension of children: $i^* \times m/2$ and $(n-i^*) \times m/2$
- local work at children:
  $c \cdot i^* \cdot m/2 + c \cdot (n-i^*) \cdot m/2 = c \cdot (i^* + (n-i^*)) \cdot m/2 = \frac{1}{2} c \cdot n \cdot m$
- Total: $O(nm)$

A Little Bio

DNA

- String over $\{A, C, G, T\}$
- Complementary strands: $A \sim T$ and $C \sim G$

RNA

- String over $\{A, C, G, U\}$
- Single strand
- Self-stabilizes forming bonds $A \sim U$ and $C \sim G$

RNA Secondary Structure

Input:
string $R[1, \ldots, n]$ over alphabet $\{A, C, G, U\}$

Output:
set S of pairs $(i, j) \in [n] \times [n]$ with $i < j$ of maximum size $|S|$ s.t.:

- [Matching] Each $i \in [n]$ appears in at most one pair of S.
- [Complementarity] For each $(i, j) \in S$, $R[i] \sim R[j]$.
- [No sharp turns] For each $(i, j) \in S$, $j \geq i + 5$.
- [No crossings] For no $(i, j), (k, \ell) \in S$, $i < k < j < \ell$.

Algorithm

Principle of optimality

- Case position 1 is not matched:
  Remains to solve problem for $R[2, \ldots, n]$.
- Case position 1 is matched with k
  (only an option if $k \geq 5$ and $R[1] \sim R[k]$):
  Remains to solve problem for $R[2, \ldots, k-1]$ and for $R[k+1, \ldots, n]$.

Subproblem specification

$\mathrm{OPT}(i, j) = \mathrm{OPT}(R[i, \ldots, j])$ where $1 \leq i \leq j \leq n$.

Recurrence (for $i < j$)

$\mathrm{OPT}(i, j) = \max(\mathrm{OPT}(i+1, j),$
$\max_{i+5 \leq k \leq j, R[i] \sim R[k]} (1 + \mathrm{OPT}(i+1, k-1) + \mathrm{OPT}(k+1, j)))$

Base cases: $\mathrm{OPT}(i, i) = 0$ for $i \in [n]$. **Answer:** $\mathrm{OPT}(1, n)$.

Analysis

Subproblem specification

$\mathrm{OPT}(i, j) = \mathrm{OPT}(R[i, \ldots, j])$ where $1 \leq i \leq j \leq n$.

Recurrence (for $i < j$)

$\mathrm{OPT}(i, j) = \max(\mathrm{OPT}(i+1, j),$
$\max_{i+5 \leq k \leq j, R[i] \sim R[k]} (1 + \mathrm{OPT}(i+1, k-1) + \mathrm{OPT}(k+1, j)))$

Time

- $\Theta(n^2)$ table entries
- $O(n)$ operations to evaluate recurrence for a given table entry
- $O(n^3)$ time overall

Space

$O(n^2)$ with or without retrieval.

Shortest Paths Problem

Input: (di)graph $G = (V, E)$; lengths $\ell : E \to \mathbb{R}$; $s, t \in V$
Ouput: path P from s to t with minimum length $\ell(P) \doteq \sum_{e \in P} \ell(e)$

Variants based on source/target

- single pair
- single source
- single target
- all pairs

Variants based on edge lengths

- unit
- nonnegative
- arbitrary

Shortest Paths Problem

Specification

Input: (di)graph $G = (V, E)$; lengths $\ell : E \to \mathbb{R}$; $s, t \in V$

Ouput: path P from s to t with minimum length $\ell(P) \doteq \sum_{e \in P} \ell(e)$

Distance $d(s,t)$

$= \min\{\ell(P) \mid P \text{ path from } s \text{ to } t\}$
$= \infty$ if there is no path from s to t
$= -\infty$ if there is a path from s to t but no shortest one

Proposition

$d(s,t) = -\infty \Leftrightarrow$
there exists cycle C with $\ell(C) < 0$ such that $s \rightsquigarrow C$ and $C \rightsquigarrow t$.

Single Source – subproblems and recurrence

Subproblems

$\mathrm{OPT}(k, v) =$
length of a shortest path from s to v using $\leq k$ edges
∞ if no such path exists
($k \in \mathbb{N}$ and $v \in V$)

Base case ($k = 0$)

$\mathrm{OPT}(0, s) = 0$ and $\mathrm{OPT}(0, v) = \infty$ for $v \neq s$

Recursive case ($k \geq 1$)

$\mathrm{OPT}(k, v) = \min($
$\min_{(u,v) \in E}(\mathrm{OPT}(k-1, u) + \ell(u,v))$
$\mathrm{OPT}(k-1, v))$

Answer

$d(s, v) = \lim_{k \to \infty} \mathrm{OPT}(k, v)$

Single Source – number of Iterations

Observation 1

If $(\forall v)\, \mathrm{OPT}(k, v) = \mathrm{OPT}(k-1, v)$
then $(\forall v)\, \mathrm{OPT}(k+1, v) = \mathrm{OPT}(k, v)$.

Observation 2

If $\mathrm{OPT}(n, u) < \mathrm{OPT}(n-1, u)$ then
there exists cycle C with $\ell(C) < 0$ such that $s \rightsquigarrow C$ and $C \rightsquigarrow u$.

Criterion

$d(s, v) = -\infty \Leftrightarrow$
there exist $u \in V$ s.t. $u \rightsquigarrow v$ and $\mathrm{OPT}(n, u) < \mathrm{OPT}(n-1, u)$.

Single Source – algorithm

```
k ← 0
OPT(k, s) ← 0
for v ∈ V \ {s} do OPT(k, v) ← ∞
repeat
    k ← k + 1
    for v ∈ V do
        d ← min_{(u,v)∈E}(OPT(k − 1, u) + ℓ(u, v)))
        OPT(k, v) ← min(OPT(k − 1, v), d)
until k = |V| or (∀v ∈ V) OPT(k, v) = OPT(k − 1, v)
for v ∈ V do
    if (∃u ∈ V) u ⇝ v and OPT(k, u) < OPT(k − 1, u) then
        d(s, v) ← −∞
    else
        d(s, v) ← OPT(k, v)
```

Single Source – complexity analysis

Recurrence

- $\mathrm{OPT}(0, s) = 0$ and $\mathrm{OPT}(0, v) = \infty$ for $v \neq s$
- $\mathrm{OPT}(k, v) =$
  $\min\left(\mathrm{OPT}(k-1, v), \min_{(u,v) \in E}(\mathrm{OPT}(k-1, u) + \ell(u, v))\right)$

Time

- $O(n+m)$ per row
- $\leq n+1$ rows
- $O(n(n+m))$ total

Space

- For computing distances: $O(n)$
- For finding shortest paths: $O(n^2) \to O(n)$

All Pairs – subproblems

Subproblems

$\mathrm{OPT}(s, t, k) =$
length of a shortest path from s to t
that only uses $\{k, \ldots, n\}$ as intermediate vertices
∞ if no such path exists
$-\infty$ if there is such a path but no shortest one
($s, t \in V \doteq [n]$ and $k \in [n+1]$)

Base case ($k = n+1$)

- $\mathrm{OPT}(s, t, n+1) = 0$ if $s = t$
- $\mathrm{OPT}(s, t, n+1) = \ell(s, t)$ if $(s, t) \in E$
- $\mathrm{OPT}(s, t, n+1) = \infty$ otherwise

Answer

$d(s, t) = \mathrm{OPT}(s, t, 1)$

All Pairs – recurrence and analysis

Recursive case ($k \leq n$)

- $\text{OPT}(s, t, k) =$
  $\min(\text{OPT}(s, t, k+1), \text{OPT}(s, k, k+1) + \text{OPT}(k, t, k+1))$
- $\text{OPT}(s, t, k) = -\infty$
  if $\text{OPT}(s, k, k+1) < \infty$, $\text{OPT}(k, t, k+1) < \infty$, and
  $\text{OPT}(k, k, k+1) < 0$

Time

- $O(n^3)$ entries
- $O(1)$ time per entry
- $O(n^3)$ time total

Space

$O(n^2)$ for distances and shortest paths