Brute Force Algorithms

CS3230: Design and Analysis of Algorithms

Roger Zimmermann

National University of Singapore

Spring 2017

Chapter 3: Brute Force

Topics: What we will cover today.

- Introduction
- Selection sort and bubble sort
- Sequential search and brute-force string matching
- Closest-pair and convex-hull problems by brute force
- Exhaustive search

The Brute Force Design Strategy

- The strategy of brute force is straightforward, or even naive, approach to solving a problem – just do it! It is usually a direct translation of the problem statement and definitions of the concepts involved.
- The advantages are:
  - Generality – works well for some problems, and if an algorithm is needed quickly.
  - Usually efficient enough for **small** input sizes.
  - An important theoretical and educational tool.

Brute Force: Example

- In cryptanalysis, a brute force attack is a method of defeating a cryptographic scheme by systematically trying a large number (i.e., exhaustive number) of possibilities.
- The design goal, on the other hand, of a cryptographic system is that it can only be defeated by brute force attack. Furthermore, the design should be such that a brute force search is computationally infeasible to carry out.
- Example: For symmetric-key ciphers, a brute force attack usually means an exhaustive search of the key space.
  - Let's say the key size is 64 bits. Then there are 2^{64} possible keys. Q: How many keys need to be tried on average to find the right key?

Selection Sort: The Idea

- Given a sequence of n keys, partition the sequence into a prefix of p keys and a suffix of s keys, $p + s = n$.
- The keys in the prefix are already in their rightful positions.
- Find the smallest key K in the suffix. Remove K from the suffix and append it to the prefix.
- Initially, $p = 0$ and $s = n$. Finally, $p = n$ and $s = 0$.

Selection Sort: An Algorithm

SELECTIONSORT
// Input: array $A[0..n-1]$
for $i \leftarrow 0$ to $n-2$ do
    $min \leftarrow i$
    for $j \leftarrow i+1$ to $n-1$ do
        if $A[j] < A[min]$ then $min \leftarrow j$ fi
    od
    swap $A[i]$ and $A[min]$
od

// suffix $A[i..n-1]$

Selection Sort: An Example

89	45	68	90	29	34	17
17	| 45	68	90	29	34	89
17	29	| 68	90	45	34	89
17	29	34	| 90	45	68	89
17	29	34	45	| 90	68	89
17	29	34	45	68	| 90	89
17	29	34	45	68	89	90

Selection Sort: An Analysis

- Let $C(n)$ be the number of key comparisons.
- Note that $C(n)$ depends only on n so there is no need to distinguish among the three cases: worst, average, best.

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1]$$

$$C(n) = \sum_{i=0}^{n-2} (n-1-i) = \frac{(n-1)n}{2} \in \Theta(n^2)$$

Selection Sort: Properties

- Runtime, average case: $O(n^2)$
- Runtime, worst case: $O(n^2)$
- Space efficiency: in-place, $O(1)$
- Stability: no (example: consider $[2_a, 2_b, 1]$)

Sorting Stability

- Stable sorting algorithms maintain the relative order of records with equal keys.
- A sorting algorithm is stable if whenever there are two records R and S with the same key, and R appears before S in the original list, then R will always appear before S in the sorted list.

- Note: When equal elements are indistinguishable, such as with integers, or more generally, any data where the entire element is the key, stability is not an issue.
- Note: Unstable sorting algorithms can usually be specially implemented to be stable.

Bubble Sort: The Idea

- Given a sequence of n keys, partition the sequence into a prefix of p keys and a suffix of s keys, $p + s = n$.
- The keys in the suffix are already in their rightful positions.
- Compare every two adjacent keys in the prefix and exchange them if they are out of order.
- In each iteration the prefix is reduced by one key and the suffix is expanded by one key at its beginning.
- Initially, $p = n$ and $s = 0$. Finally, $p = 0$ and $s = n$.

Bubble Sort: An Algorithm

BUBBLESORT
// Input: array $A[0..n-1]$
for $i \leftarrow 0$ to $n-2$ do
    for $j \leftarrow 0$ to $n-2-i$ do
        if $A[j+1] < A[j]$ then
            swap $A[j]$ and $A[j+1]$
        fi
    od
od

Bubble Sort: An Example

89	$\overset{?}{\leftrightarrow}$	45		68		90		29		34		17
45		89	$\overset{?}{\leftrightarrow}$	68		90		29		34		17
45		68		89	$\overset{?}{\leftrightarrow}$	90	$\overset{?}{\leftrightarrow}$	29		34		17
45		68		89		29		90	$\overset{?}{\leftrightarrow}$	34		17
45		68		89		29		34		90	$\overset{?}{\leftrightarrow}$	17
45		68		89		29		34		17	|	90
45	$\overset{?}{\leftrightarrow}$	68	$\overset{?}{\leftrightarrow}$	89	$\overset{?}{\leftrightarrow}$	29		34		17	|	90
45		68		29		89	$\overset{?}{\leftrightarrow}$	34		17	|	90
45		68		29		34		89	$\overset{?}{\leftrightarrow}$	17	|	90
45		68		29		34		17	|	89		90

Bubble Sort: An Analysis

- Let $C(n)$ be the number of key comparisons.
- Note that $C(n)$ depends only on n so there is no need to distinguish among the three cases: worst, average, best. (Key swaps depend on input.)

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=0}^{n-2-i} 1 = \sum_{i=0}^{n-2} [(n-2-i) - 0 + 1]$$

$$C(n) = \sum_{i=0}^{n-2} (n-1-i) = \frac{(n-1)n}{2} \in \Theta(n^2).$$

Bubble Sort: Summary

- Runtime, average case: $O(n^2)$
- Runtime, worst case: $O(n^2)$
- Space efficiency: in-place, $O(1)$
- Stability: yes

Sequential Search with A Sentinel: The Idea

- A brute force sequential search looks for a key among a sequence of keys one after another.
- The search stops when a match is found or the search goes beyond the sequence.
- Going beyond the sequence is prevented by appending the search key to the end of the sequence.

Sequential Search with A Sentinel: An Algorithm

- The algorithm:

  SEQUENTIALSEARCH
  // Input: array $A[0..n]$; $A[n]$ is the sentinel
  // The target key is K
  $A[n] \leftarrow K$
  $i \leftarrow 0$
  while $A[i] \neq K$ do $i \leftarrow i + 1$ od
  if $i < n$ return i
  else return -1 fi

- Clearly, the number of comparisons $C(n) = n + 1$ in the worst case.

Brute Force String Matching: The Idea

- The problem of string matching is about finding a length m substring (the pattern) among a length n string (the text).
- This can be done by checking the $n - m + 1$ substrings starting at the text positions $0, 1, \cdots, n - m$.

Brute Force String Matching: An Algorithm

Step 1: Align pattern at the beginning of text.

Step 2: Moving from left to right, compare each character of pattern to the corresponding character in text until

- all characters are found to match (successful search); or
- a mismatch is detected.

Step 3: While pattern is not found and the text is not yet exhausted, realign pattern one position to the right and repeat Step 2.

Brute Force String Matching: An Algorithm

```
StringMatching
// Input text: T[0..n − 1]
// Input pattern: P[0..m − 1]
for i ← 0 to n − m do
    j ← 0
    while j < m and P[j] = T[j + i] do
        j ← j + 1
    od
    if j = m then return i fi
od
return −1
```

Brute Force String Matching: Examples

1. Pattern: 001011

   Text: $10010101101001100101111010$

2. Pattern: *happy*

   Text: *It is never too late to have a happy childhood*.

Brute Force String Matching: An Analysis

- Let $C(n)$ be the number of comparisons $P[j] = T[j + i]$.
- For the worst case,

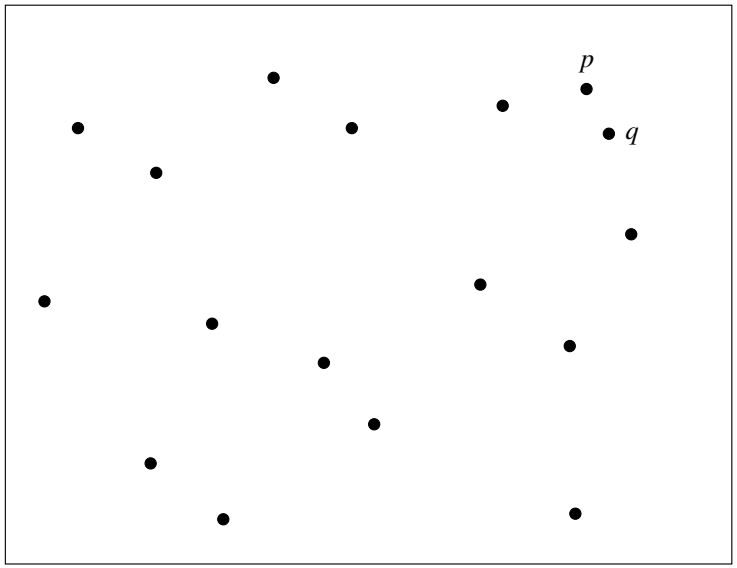
$$C(n) \leq (n - m + 1)m \in O(nm).$$

- It can be shown that the worst case $C(n) \in \Theta(nm)$.

- It has been shown that for random text the average-case efficiency is linear: $C(n) \in \Theta(n + m) = \Theta(n)$.

Closest Pair Problem: The Idea

- The closest pair problem seeks a pair of points which are closest among a set of n points (in m-dimensions).
- This is a fundamental problem in many applications as well as a key step in many algorithms.
- Example (2D):



Closest Pair Problem: 2D Space

- The closest pair problem can be formulated in 1-, 2- and m-dimensional space. Here we consider the 2D problem.
- Points are specified by their (x, y) Cartesian coordinates.
- The distance between two points P_1 and P_2 is the standard Euclidean distance:

$$d(P_i, P_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}.$$

- Consider only $i < j$. (Q: Why?)
- A brute force algorithm simply checks all the $\binom{n}{2}$ pairs of points.
- We will find a better divide-and-conquer algorithm later.

Closest Pair Problem: An Algorithm

CLOSESTPAIR
// Input $x[1..n], y[1..n], P[i] = \langle x[i], y[i] \rangle$
$d_{min} \leftarrow \infty$
for $i \leftarrow 1$ to $n - 1$ do
    for $j \leftarrow i + 1$ to n do
        $d \leftarrow (x[i] - x[j])^2 + (y[i] - y[j])^2$
        if $d < d_{min}$ then
            $d_{min} \leftarrow d$; $i_1 \leftarrow i$; $i_2 \leftarrow j$
        fi
    od
od
return i_1, i_2

Closest Pair Problem: An Analysis

- Q: Why is there no square-root function in the algorithm?

- Q: Can we also remove the $(\ldots)^2$ squaring operations? For example, by using the absolute $|\ldots|$ values?

Closest Pair Problem: An Analysis (2)

- Let $C(n)$ be the number of times a number is squared: $(\ldots)^2$.
- Clearly $C(n)$ depends on n only:

$$C(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2 = 2 \sum_{i=1}^{n-1} (n-i)$$

$$C(n) = 2[(n-1) + (n-2) + \cdots + 1] = (n-1)n \in \Theta(n^2).$$

- A brute force algorithm for m-dimensions takes $O(mn^2)$.
- We will later find an $O(n \log n)$ algorithm (2D).

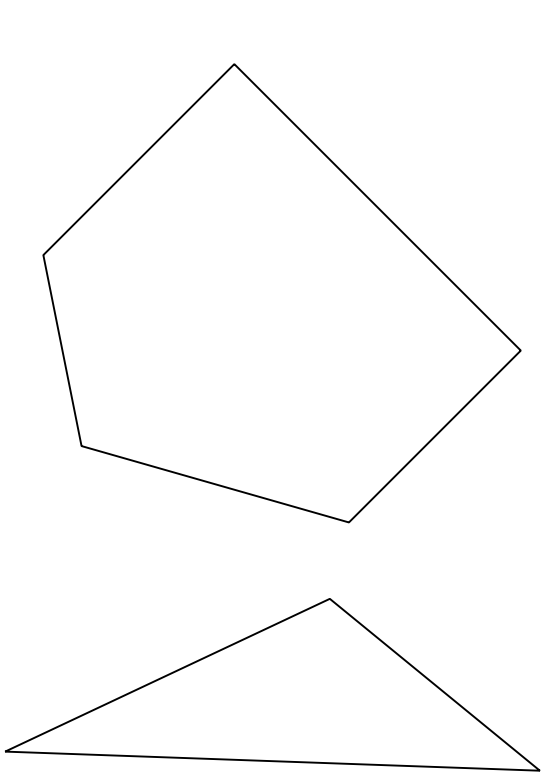
Convex Hull

- A shape S is convex if for any points $P, Q \in S$, the line segment PQ is contained in S:

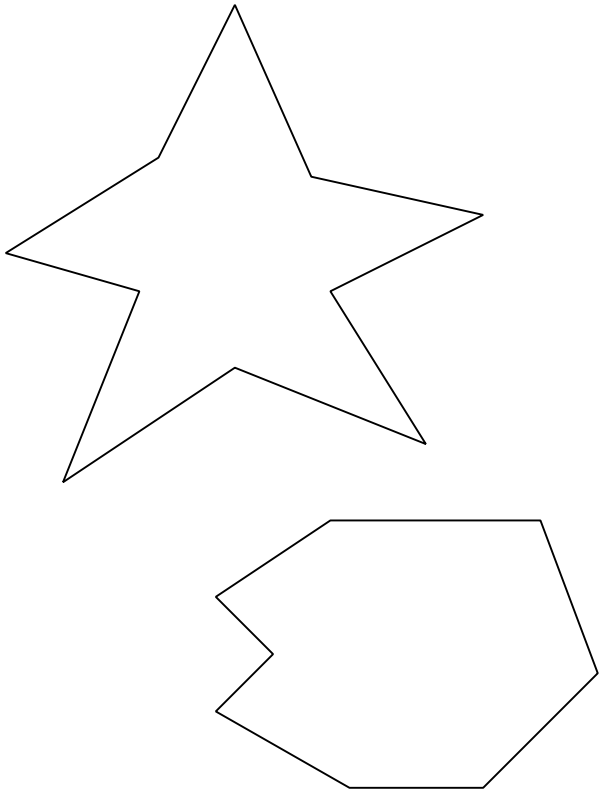
$$PQ = \{x | x = (1 - \lambda)P + \lambda Q, 0 \leq \lambda \leq 1\} \subseteq S.$$

- Given an arbitrary shape S, the convex hull conv(S) of S is the smallest convex shape that contains S.
- That is,
  1. conv(S) is convex; and
  2. for any convex $S' \supseteq S$, conv(S) $\subseteq S'$.

Convex Set Examples



Convex sets



Non-convex sets

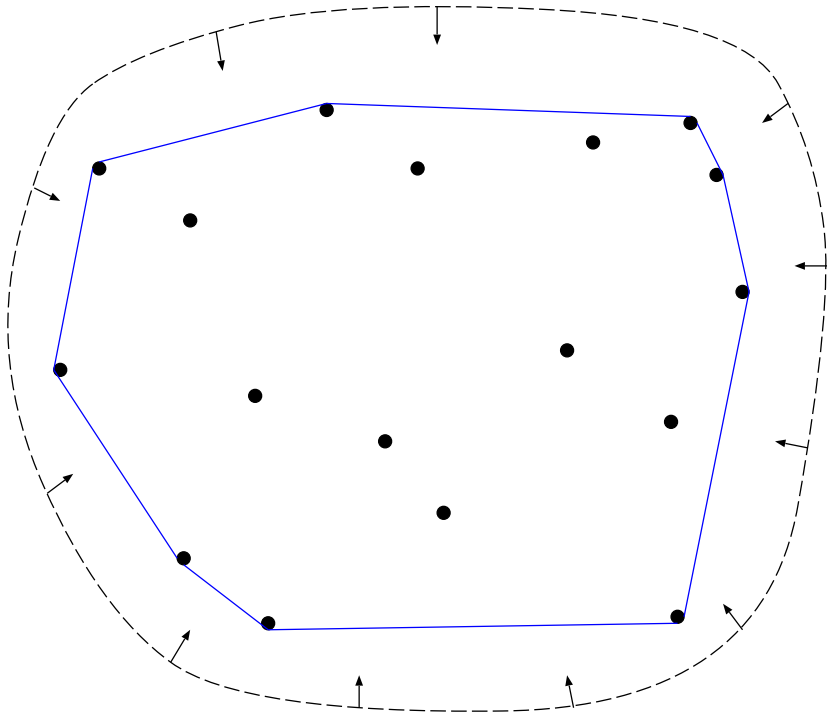
A Theorem on Convex Hulls

- THEOREM The convex hull of a finite set S of points is a convex polygon whose vertices are points of the given set S.

- To find a convex hull of a set of points we need to find extreme points of the set.

A Theorem on Convex Hulls: Example

- Intuitive picture: a convex hull may be visualized by imagining an elastic band stretched open to encompass all the given points. (For planar objects.)



Convex Hull Applications

- Examples: GIS (Geographical Information Systems), visual communications, visibility determination.

A Theorem from Analytic Geometry

- THEOREM The line $ax + by + c = 0$ partitions the plane (x, y) into three parts:

$$ax + by + c < 0, \quad ax + by + c = 0, \quad ax + by + c > 0.$$

- Tip: A line segment connecting two points P_i and P_j of a set of n points is a part of its convex hull's boundary if and only if all the other points of the set lie on the same side of the straight line through these two points.

A Brute Force Convex Hull Algorithm

CONVEXHULL
// Input: n points $P[0..n-1]$
// $V[0..n-1]$ is an auxiliary array initialized to 0
for $i \leftarrow 0$ to $n-2$ do
    for $j \leftarrow i+1$ to $n-1$ do
        find the line equ. $f(x,y)=0$ passing thru $P[i], P[j]$
        find the sign of $f(P[k]), k=0,...,n-1; k \neq i,j$
        if all signs are the same then set $V[i], V[j]$ to 1
    od
od

- Output: The convex hull is the polygon whose vertices are $P[i]$ with $V[i]=1$.

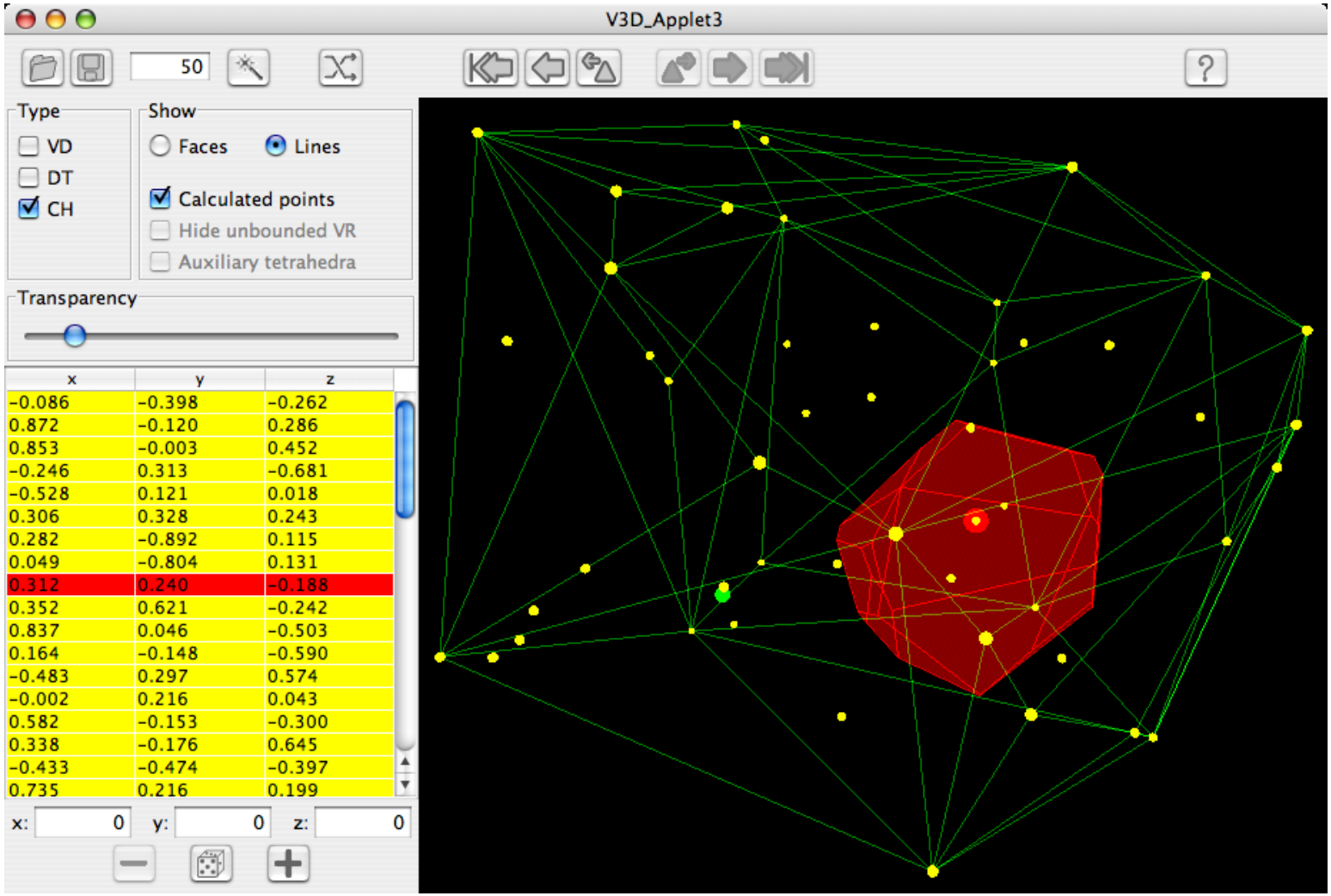
A Brute Force Convex Hull Algorithm: Analysis

- Let $C(n)$ be the number of times a line equation $f(x, y)$ is evaluated.
- Clearly $C(n)$ depends only on n.
- We have

$$C(n) \leq \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} (n-2) = \frac{n(n-1)(n-2)}{2} \in \Theta(n^3).$$

- Note: For each of the $n(n-1)/2$ pairs of distinct points we may need to find the sign of $ax + by + c$ for each of the other $n - 2$ points.

Convex Hull: 3D Example



Convex hull of 50 randomly generated points and one highlighted bounded Voronoi region.

http://www.voronoi3d.com/index.html

Exhaustive Search

- Exhaustive search is a brute force approach to combinatorial problems.
- Given a problem, an exhaustive search algorithm generates all possible candidates for the problem.
- Among the possible candidates the algorithm selects those that satisfy some objects.
- For example, with small input sizes, exhaustive search solves the traveling salesman problem, the knapsack problem, and the assignment problem easily.

The Traveling Salesman Problem

- The problem:

  Given a set of cities connected by roads, find a simple cycle that includes all the cities with the shortest total distance.

- An exhaustive algorithm:
  1. Label the n cities from 0 to $n-1$.
  2. Generate the $(n-1)!$ permutations P of the integers from 1 to $n-1$. (Q: Why start from 1?)
  3. For each $p \in P$, check that p is a path and compute the weight $w(p)$.
  4. Report $q \in P$ that has the smallest weight $w(q)$.

The Traveling Salesman Problem: Example

- Consider the weighted graph:

	a	b	c	d
a		2	5	7
b	2		8	3
c	5	8		1
d	7	3	1	

- $\exists\ 6 = 3!$ cycles starting at a:

$abcda$	$2 + 8 + 1 + 7 = 18$
$abdca$	$2 + 3 + 1 + 5 = 11*$
$acbda$	$5 + 8 + 3 + 7 = 23$
$acdba$	$5 + 1 + 3 + 2 = 11*$
$adbca$	$7 + 3 + 8 + 5 = 23$
$adcba$	$7 + 1 + 8 + 2 = 18$

The Knapsack Problem

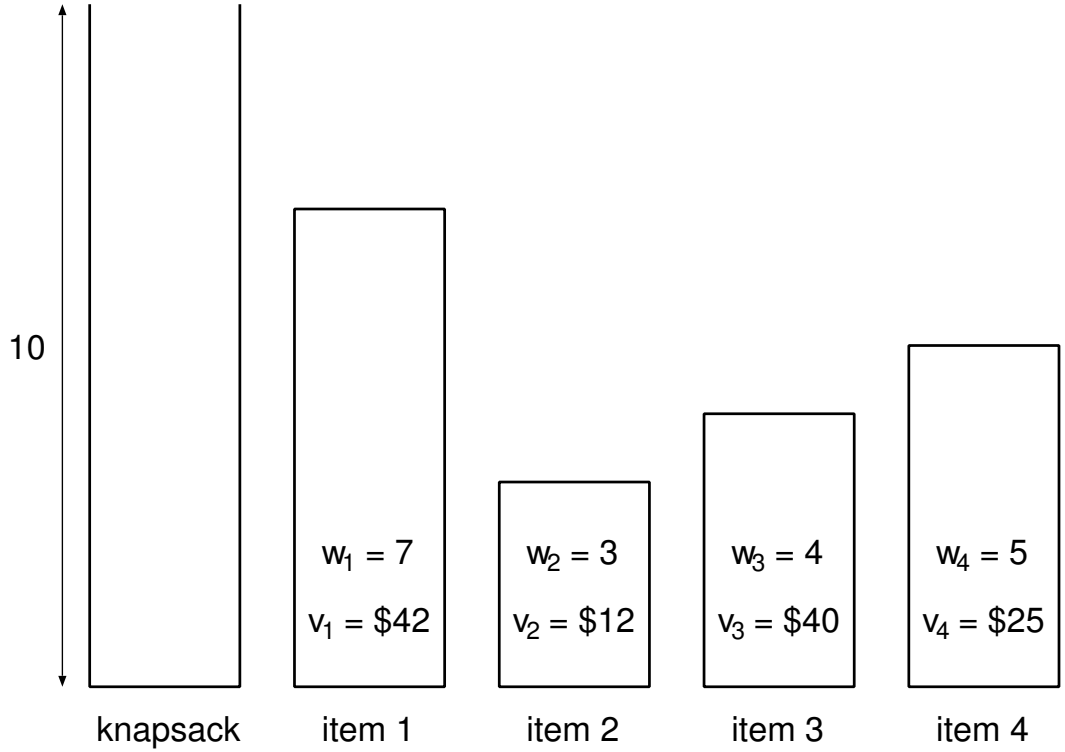
- The problem:

  There are n items, each has a value and a weight. There is a knapsack with capacity W. Fill the knapsack with items such that the capacity is not exceeded and the total value is largest.

- An exhaustive search solution:
  1. Generate all the $2^n - 1$ non-empty subsets of the given set of n items.
  2. For each subset, compute the total weight and total value.
  3. Find the largest total value among those subsets whose total weight does not exceed W and report it.

The Knapsack Problem: Example

- A knapsack has capacity 10. There are 4 items with attribute (weight, value) given as
$A_1 = (7, 42), A_2 = (3, 12), A_3 = (4, 40), A_4 = (5, 25)$.



The Knapsack Problem

Subset	Total weight	Total value
$\emptyset$	0	\$0
$\{1\}$	7	\$42
$\{2\}$	3	\$12
$\{3\}$	4	\$40
$\{4\}$	5	\$25
$\{1,2\}$	10	\$36
$\{1,3\}$	11	not feasible
$\{1,4\}$	12	not feasible
$\{2,3\}$	7	\$52
$\{2,4\}$	8	\$37
$\{3,4\}$	9	\$65
$\{1,2,3\}$	14	not feasible
$\{1,2,4\}$	15	not feasible
$\{1,3,4\}$	16	not feasible
$\{2,3,4\}$	12	not feasible
$\{1,2,3,4\}$	19	not feasible

The Knapsack Problem: Example

- A knapsack has capacity 10. There are 4 items with attribute (weight, value) given as $A_1 = (7, 42), A_2 = (3, 12), A_3 = (4, 40), A_4 = (5, 25)$.
- Among the 15 non-empty subsets of the set $\{1, 2, 3, 4\}$, the 7 subsets $\{1, 3\}, \{1, 4\}, \{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}, \{2, 3, 4\}, \{1, 2, 3, 4\}$ exceed the knapsack capacity.
- The 8 possible candidates are $\{1\}, \{2\}, \{3\}, \{4\}, \{1, 2\}, \{2, 3\}, \{2, 4\}, \{3, 4\}$.
- The candidate $\{3, 4\}$ gives the largest value of 65.

The Assignment Problem

- The problem:

  There are n jobs to be assigned to n persons. The cost of person i doing job j is $C[i, j]$. Find an assignment that has the lowest cost.

- An exhaustive search solution:

  1. Generate $n!$ permutations of the integers $1, \cdots, n$.
  2. For the permutation $i_1, \cdots, i_n$, compute the assignment cost $\sum_{j=1}^{n} C[i, j]$.
  4. Report an assignment with the smallest total cost.

The Assignment Problem: Example

- Let the cost matrix be

	J1	J2	J3	J4
P1	9	2	7	8
P2	6	4	3	7
P3	5	8	1	8
P4	7	6	9	4

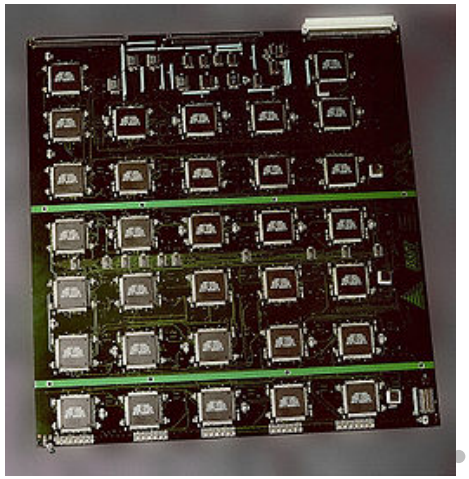
- There are $4! = 24$ possible assignments.
- As an example, for the assignment 1432, the assignment cost is $C[1,1] + C[2,4] + C[3,3] + C[4,2] = 9 + 7 + 1 + 6 = 23$.
- It can be checked that the permutation 2134 gives the smallest assignment cost $2 + 6 + 1 + 4 = 13$.

Brute Force Search: Example

- In cryptanalysis, a brute force attack is a method of defeating a cryptographic scheme by systematically trying a large number (i.e., exhaustive number) of possibilities.
- The design goal, on the other hand, of a cryptographic system is that it can only be defeated by brute force attack. Furthermore, the design should be such that a brute force search is computationally infeasible to carry out.
- Example: For symmetric-key ciphers, a brute force attack usually means an exhaustive search of the key space.
  - Let's say the key size is 64 bits. Then there are 2^{64} possible keys. How many keys need to be tried on average to find the right key?

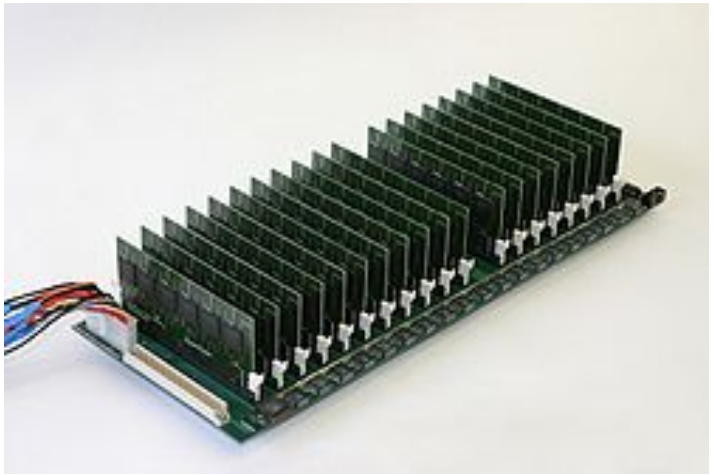
Brute Force Special Hardware

- One way to speed up a brute force search is to parallelize the search operations through the search space.
- Examples:
  - The Electronic Frontier Foundation (EFF) DES cracker is a machine built in 1998 to perform a brute force search of DES cipher's key space. It cost US$250,000 to build and contained 1,856 custom chips. It could crack a DES key in a matter of days.



Brute Force Special Hardware (2)

- DES: Data Encryption Standard (designed by IBM). A block cipher, selected in 1976 by US NIST. It uses a symmetric key of length 56 bits.
- Examples:
  - The COPACABANA machine is a reprogrammable hardware based on FPGAs. It was built for US$10,000 in 2006 and it contained 120 chips.



Brute Force Special Hardware (3)

- Q: Can a cipher with a key length 128 bits (or 256 bits) be realistically searched brute force?

- Q: One assumption for cryptograpy is that a key is generated truly randomly. What happens if that is not the case?

Take Away Message on Exhaustive Search

- Exhaustive-search algorithms run in a realistic amount of time only on very small instances.
- In some cases, there are much better alternatives!
  - Euler circuits
  - shortest paths
  - minimum spanning tree
  - assignment problem
- In many cases, exhaustive search or its variations are the only known way to get exact solutions.