Combinatorial Search

- subsets
- permutations
- counting
- paths in a graph
- paths in a lattice

subsets
permutations
counting
paths in a lattice
path in a graph

Overview

Exhaustive search. Iterate through all elements of a search space.

Backtracking. Systematic method for examining feasible solutions to a problem, by systematically eliminating infeasible solutions.

Applicability. Huge range of problems (include NP-hard ones).

Caveat. Search space is typically exponential in size ⇒ effectiveness may be limited to relatively small instances.

Caveat to the caveat. Backtracking may prune search space to reasonable size, even for relatively large instances

Enumerating subsets: natural binary encoding

Given n items, enumerate all 2^n subsets.
- count in binary from 0 to $2^n - 1$.
- bit i represents item i
- if 0, in subset; if 1, not in subset

i	binary	subset	complement
0	0 0 0 0	empty	4 3 2 1
1	0 0 0 1	1	4 3 2
2	0 0 1 0	2	4 3 1
3	0 0 1 1	2 1	4 3
4	0 1 0 0	3	4 2 1
5	0 1 0 1	3 1	4 2
6	0 1 1 0	3 2	4 1
7	0 1 1 1	3 2 1	4
8	1 0 0 0	4	3 2 1
9	1 0 0 1	4 1	3 2
10	1 0 1 0	4 2	3 1
11	1 0 1 1	4 2 1	3
12	1 1 0 0	4 3	2 1
13	1 1 0 1	4 3 1	2
14	1 1 1 0	4 3 2	1
15	1 1 1 1	4 3 2 1	empty

Enumerating subsets: natural binary encoding

Given n items, enumerate all 2^n subsets.

- count in binary from 0 to 2^n - 1.
- bit i represents item i
- if 0, in subset; if 1, not in subset

```
long N = 1 << n;
for (long i = 0; i < N; i++)
{
   for (int bit = 0; bit < n; bit++)
   {
      if (((i >> bit) & 1) == 1)
         System.out.print(bit + " ");
   }
   System.out.println();
}
```

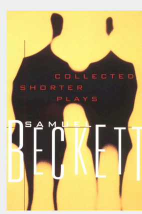
Note: bitflicking simpler in assembly language

Samuel Beckett

Quad. Starting with empty stage, 4 characters enter and exit one at a time, such that each subset of actors appears exactly once.

code	subset	move
0 0 0 0	empty	
0 0 0 1	1	enter 1
0 0 1 1	2 1	enter 2
0 0 1 0	2	exit 1
0 1 1 0	3 2	enter 3
0 1 1 1	3 2 1	enter 1
0 1 0 1	3 1	exit 2
0 1 0 0	3	exit 1
1 1 0 0	4 3	enter 4
1 1 0 1	4 3 1	enter 1
1 1 1 1	4 3 2 1	enter 2
1 1 1 0	4 3 2	exit 1
1 0 1 0	4 2	exit 3
1 0 1 1	4 2 1	enter 1
1 0 0 1	4 1	exit 2
1 0 0 0	4	exit 1

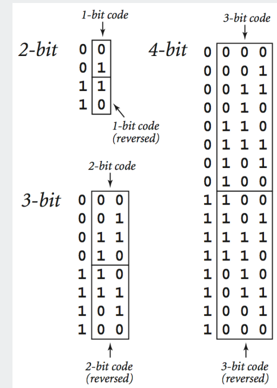
ruler function



Enumerating Subsets: Binary Reflected Gray Code

Binary reflected Gray code. The n-bit code is:

- the (n-1) bit code with a 0 prepended to each word, followed by
- the (n-1) bit code in reverse order, with a 1 prepended to each word.



Beckett: Java implementation

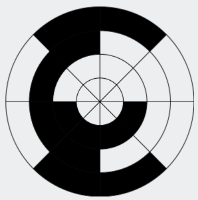
```
public static void moves(int n, boolean enter)
{
   if (n == 0) return;
   moves(n-1, true);
   if (enter) System.out.println("enter " + n);
   else       System.out.println("exit  " + n);
   moves(n-1, false);
}
```

```
% java Beckett 4
enter 1
enter 2
exit  1
enter 3
enter 1
exit  2
exit  1
enter 4
enter 1
enter 2
exit  1
exit  3
enter 1
exit  2
exit  1
```

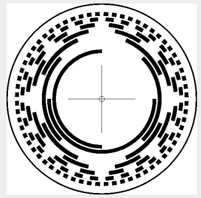
stage directions for 3-actor play `moves(3, true)`

reverse stage directions for 3-actor play `moves(3, false)`

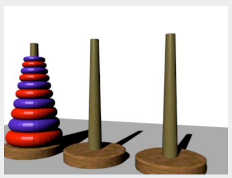
More Applications of Gray Codes



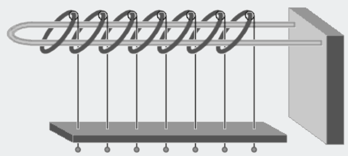
3-bit rotary encoder



8-bit rotary encoder



Towers of Hanoi



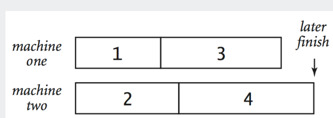
Chinese ring puzzle

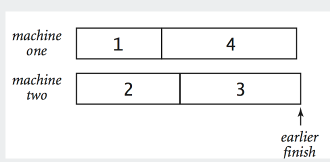
Scheduling

Scheduling (set partitioning). Given n jobs of varying length, divide among two machines to minimize the time the last job finishes.

or, equivalently, difference between finish times

job	length
1	1.41
2	1.73
3	2.00
4	2.23





Remark. NP-hard.

Scheduling (using Gray Code)

gap = sum

+2.23 if job 4 on machine one
-2.23 if job 4 on machine two

Beckett's stage directions

flip job 4 from machine one to machine two

	a[0]	a[1]	a[2]	a[3]	a[4]
	7.37	1.41	1.73	2.00	2.23
1	4.55	-1.41	1.73	2.00	2.23
2	1.09	-1.41	-1.73	2.00	2.23
1	3.91	1.41	-1.73	2.00	2.23
3	-0.09	1.41	-1.73	-2.00	2.23
1	-2.91	-1.41	-1.73	-2.00	2.23
2	0.55	-1.41	1.73	-2.00	2.23
1	3.38	1.41	1.73	-2.00	2.23
4	-1.08	1.41	1.73	-2.00	-2.23
1	-3.91	-1.41	1.73	-2.00	-2.23
2	-7.37	-1.41	-1.73	-2.00	-2.23
1	-4.55	1.41	-1.73	-2.00	-2.23
3	-0.55	1.41	-1.73	2.00	-2.23
1	-3.38	-1.41	-1.73	2.00	-2.23
2	0.09	-1.41	1.73	2.00	-2.23
1	2.91	1.41	1.73	2.00	-2.23

Scheduling: Java implementation

current schedule — best schedule so far

```
public static void moves(int n, double[] a, double[] b)
{
   if (n == 0) return;
   moves(n-1, a, b);


   a[n] = -a[n];
   a[0] += 2*a[n];
   if (Math.abs(a[0]) < Math.abs(b[0]))
      for (int i = 0; i < a.length; i++)
         b[i] = a[i];


   moves(n-1, a, b);
}
```

flip machine for job n ; fix total time

check whether schedule is the best so far

sum — job lengths

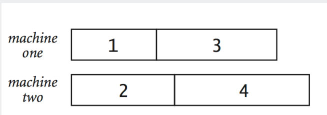
```
int[] a  = { 7.37, 1.41, 1.73, 2.00, 2.23 };

int[] b  = { 7.37, 1.41, 1.73, 2.00, 2.23 };
```

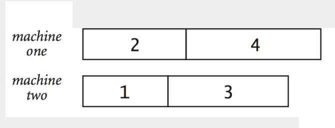
best schedule so far

Exploiting Symmetry

Exploit symmetry.

- Half of schedules are redundant.





- Fix job n on machine one $\Rightarrow$ twice as fast.

subsets
permutations
counting
paths in a lattice
paths in a graph

Space-Time Tradeoff

job	length
1	1.41
2	1.73
3	2.00
4	2.23
5	3.00
6	0.35

Space-time tradeoff.

- Enumerate all subsets of first n/2 jobs; sort by gap.

gap (subset)	-5.14 (empty)	-2.32 (1)	-1.68 (2)	-1.14 (3)	1.14 (1 2)	1.68 (1 3)	2.32 (2 3)	5.14 (1 2 3)

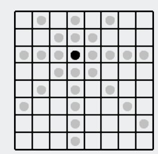
- Enumerate all subsets of last n/2 jobs; for each subset, binary search to find for best matching subset among first n/2 jobs.

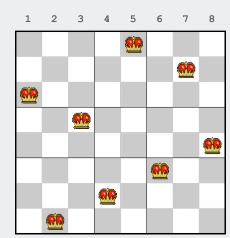
gap (subset)	-5.58 (empty)	-1.12 (4)	0.42 (5)	-4.88 (6)	4.48 (4 5)	-0.42 (4 6)	1.12 (5 6)	5.58 (4 5 6)
best match	5.14 (1 2 3)	1.14 (1 2)	-1.14 (3)	5.14 (1 2 3)	-5.14 (empty)	1.14 (2)	-1.14 (3)	-5.14 (empty)
gap	-0.44 (1 2 3)	0.02 (1 2 4)	-0.72 (3 5)	0.26 (1 2 3 6)	-0.26 (4 5)	0.72 (2 4 6)	0.02 (3 5 6)	0.44 (1 2 3)

Reduces running time from 2^n to $2^{n/2}$ log n by consuming $2^{n/2}$ memory.

8-Queens Problem

8-queens problem. Place 8 queens on a chessboard so that no queen can attack any other queen.





Representation. Can represent solution as a permutation: `q[i]` = column of queen in row `i`.

```
int[] q = { 5, 7, 1, 3, 8, 6, 4, 2 };
```

queens `i` and `j` can attack each other if `|q[i] + i| = |q[j] + j|`

Enumerating Permutations

Permutations. Given n items, enumerate all n! permutations.

3-element permutations

1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1

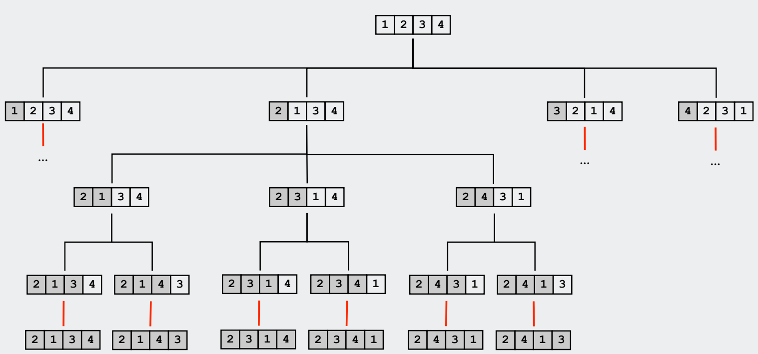
4-element permutations

1 2 3 4	2 1 3 4	3 1 2 4	3 1 2 4
1 2 4 3	2 1 4 3	3 1 4 2	3 1 4 2
1 3 2 4	2 3 1 4	3 2 1 4	3 2 1 4
1 3 4 2	2 3 4 1	3 2 4 1	3 2 4 1
1 4 2 3	2 4 1 3	3 4 1 2	3 4 1 2
1 4 3 2	2 4 3 1	3 4 2 1	3 4 2 1
1 followed by any permutation of 2 3 4	2 followed by any permutation of 1 3 4	3 followed by any permutation of 1 2 4	3 followed by any permutation of 1 2 4

Enumerating all Permutations

To enumerate all permutations of a set of n elements:

- For each element a_i
  - put a_i first, then append
  - a permutation of the remaining elements (a_0, …, a_{i-1}, a_{i+1}, …, a_{n-1})



Enumerating all permutations: Java Implementation

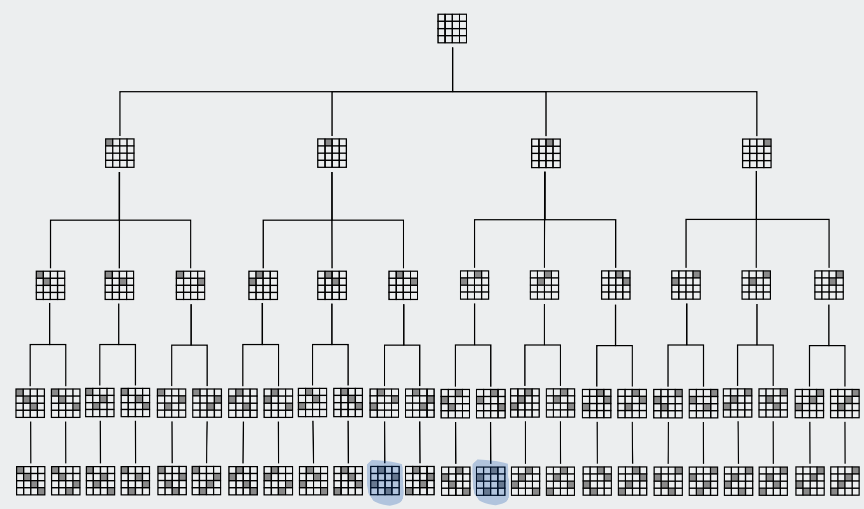
permutations of a[n], …, a[N-1]

```
private static void enumerate(int[] a, int n)
{
   int N = a.length;
   if (n == N) printPermutations(a);
   for (int i = n; i < N; i++)
   {
      swap(q, i, n);
      enumerate(a, n+1);
      swap(q, n, i);
   }
}
```

clean up

```
int N = 4;
int[] a = { 1, 2, 3, 4 };
enumerate(a, N);
```


4-Queens Search Tree



N Queens: Backtracking solution

Backtracking. Iterate through elements of search space.

- for each row, there are N possible choices.
- make one choice and recur.
- if the choice does not work, backtrack to previous choice, and make next available choice.

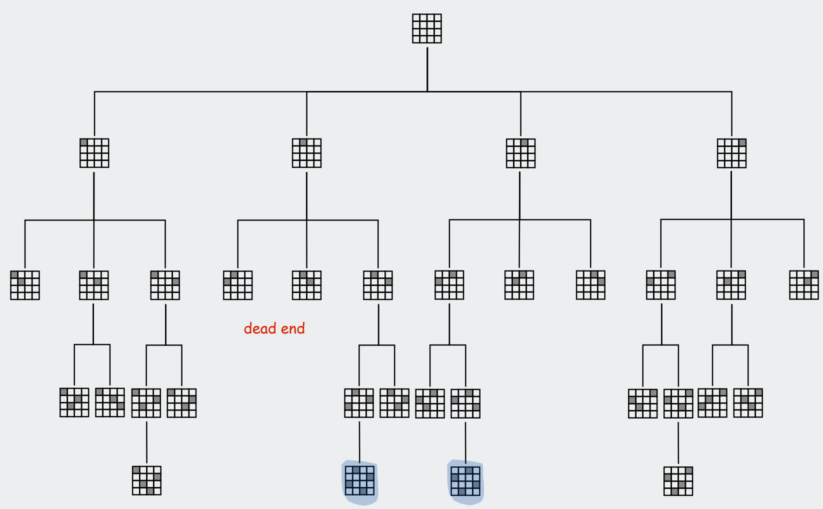
Backtracking amounts to pruning the search space.

For N queens: if you find a diagonal conflict, no need to continue

Improvements.

- try to make an "intelligent" choice
- try to reduce cost of choosing/backtracking

4-Queens Search Tree (pruned)



N-Queens: Backtracking solution

```
private static void enumerate(int[] q, int n)
{
   int N = q.length;
   if (n == N) printQueens(q);
   for (int i = n; i < N; i++)
   {
      swap(q, i, n);
      if (isConsistent(q, n)) enumerate(q, n+1);
      swap(q, n, i);
   }
}
```

stop enumerating if adding the n^{th} queen leads to a diagonal violation

```
int N = 4;
int[] q = { 1, 2, 3, 4 };
enumerate(q, N);
```


subsets
permutations
counting
paths in a lattice
paths in a graph

Counting: Java Implementation

Enumerate all M-digit base-R numbers.

```
private static void count(int[] number, int digit)
{
   if (digit == M)
   {  show(number); return;  }

   for (int n = 0; n < R; n++)
   {
      count(number, digit + 1);
   }
   number[digit] = 0;
}
```

```
0 0 0     1 0 0     2 0 0
0 0 1     1 0 1     2 0 1
0 0 2     1 0 2     2 0 2
0 1 0     1 1 0     2 1 0
0 1 1     1 1 1     2 1 1
0 1 2     1 1 2     2 1 2
0 2 0     1 2 0     2 2 0
0 2 1     1 2 1     2 2 1
0 2 2     1 2 2     2 2 2
```


Sudoku

Fill 9-by-9 grid so that every row, column, and box contains the digits 1 through 9.

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

Remark. Natural generalization is NP-hard.

Sudoku

Fill 9-by-9 grid so that every row, column, and box contains the digits 1 through 9.

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

Remark. Natural generalization is NP-hard.

Sudoku

Linearize. Treat 9-by-9 array as an array of length 81.

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

7		8					3		...	
0	1	2	3	4	5	6	7	8		80

Enumerate all assignments. Count from 0 to $9^{81} - 1$ in base 9.

using digits 1 to 9

Sudoku: Backtracking solution

Backtracking. Iterate through elements of search space.

- For each empty cell, there are 9 possible choices.
- Make one choice and recur.
- If you reach a contradiction, backtrack to previous choice, and make next available choice.

Backtracking amounts to pruning the search space.

For Sudoko:
if you find a conflict in row, column or box, no need to continue

Improvements.

- try to make an "intelligent" choice
- try to reduce cost of choosing/backtracking

subsets
permutations
counting
paths in a lattice
paths in a graph

Sudoku: Java implementation

```
private static void solve(int[] board, int cell)
{
   if (cell == 81)
   {  show(board); return;  }

   if (board[cell] != 0)
   {  solve(board, cell + 1); return;  }

   for (int n = 1; n <= 9; n++)
   {
      if (isConsistent(board, cell, n))
      {
         board[cell] = n;
         solve(board, cell + 1);
      }
   }
   board[cell] = 0;
}
```

found the solution

skip cell n since value set at initialize

try all 9 possibilities

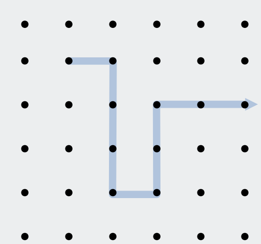
unless a Sudoku constraint is violated

cleans up after itself

```
int[] board = { 7, 0, 8, 0, 0, 0, 3, … };
solve(board, 0);
```


All Paths on a Grid

All paths. Enumerate all simple paths on a grid of adjacent sites.

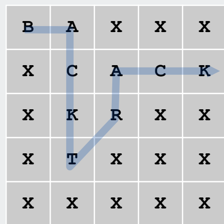


Application. Self-avoiding lattice walk to model polymer chains.

no atoms can occupy same position at same time

Boggle

Boggle. Find all words that can be formed by tracing a simple path of adjacent cubes (left, right, up, down, diagonal).



Pruning. Stop as soon as no word in dictionary contains string of letters on current path as a prefix ⇒ use a trie.

B
BA
BAX

Boggle: Java Implementation

```
private void dfs(String prefix, int i, int j)
{
   if (i < 0 || i >= N) ||
      (j < 0 || j >= N) ||
      (visited[i][j])   ||
      !dictionary.containsAsPrefix(prefix))
      return;

   visited[i][j] = true;
   prefix = prefix + board[i][j];

   if (dictionary.contains(prefix))
      found.add(prefix);

   for (int ii = -1; ii <= 1; ii++)
      for (int jj = -1; jj <= 1; jj++)
         dfs(prefix, i + ii, j + jj);

   visited[i][j] = false;
}
```

backtrack

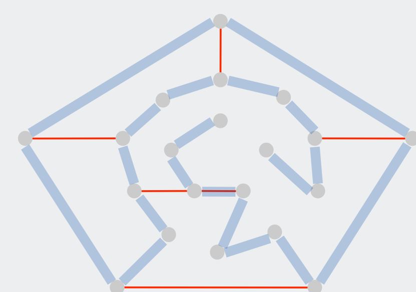
add to set of found words

clean up

- subsets
- permutations
- counting
- paths in a lattice
- **paths in a graph**

Hamilton Path

Hamilton path. Find a simple path that visits every vertex exactly once.

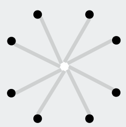


Remark. Euler path easy, but Hamilton path is NP-complete.

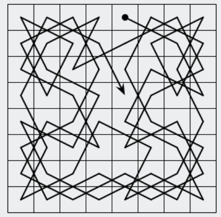
visit every edge exactly once

Knight's Tour

Knight's tour. Find a sequence of moves for a knight so that, starting from any square, it visits every square on a chessboard exactly once.



legal knight moves



a knight's tour

Solution. Find a Hamilton path in knight's graph.

Hamilton Path: Backtracking Solution

Backtracking solution. To find Hamilton path starting at `v`:

- Add `v` to current path.
- For each vertex `w` adjacent to `v`
  - find a simple path starting at `w` using all remaining vertices
- Remove `v` from current path.

How to implement?

- add cleanup to DFS (!)

Hamilton Path: Java implementation

```
public class HamiltonPath
{
   private boolean[] marked;
   private int[] pred;

   public HamiltonPath(Graph G)
   {
      marked = new boolean[G.V()];
      for (int v = 0; v < G.V(); v++)
         dfs(G, v, 1);
   }

   private void dfs(Graph G, int v, int depth)
   {
      marked[v] = true;

      if (depth == G.V()) StdOut.println("Path found!");

      for (int w : G.adj(v))
         if (!marked[w])
            { pred[w] = v; dfs(G, w); }

      marked[v] = false;
  }
}
```


The Longest Path

Recorded by Dan Barrett in 1988 while a student at Johns Hopkins during a difficult algorithms final.

Woh-oh-oh-oh, find the longest path!
Woh-oh-oh-oh, find the longest path!

If you said P is NP tonight,
There would still be papers left to write,
I have a weakness,
I'm addicted to completeness,
And I keep searching for the longest path.

The algorithm I would like to see
Is of polynomial degree,
But it's elusive:
Nobody has found conclusive
Evidence that we can find a longest path.

I have been hard working for so long.
I swear it's right, and he marks it wrong.
Some how I'll feel sorry when it's done: GPA 2.1
Is more than I hope for.

Garey, Johnson, Karp and other men (and women)
Tried to make it order N log N.
Am I a mad fool
If I spend my life in grad school,
Forever following the longest path?

Woh-oh-oh-oh, find the longest path!
Woh-oh-oh-oh, find the longest path!
Woh-oh-oh-oh, find the longest path.

Have a good summer!

Course evaluation info

Course. COS 226
Term. Spring '07.
Lecturer. Robert Sedgewick
Precept instructor. Jimin Song (01)
or David Walker (01A or 02)
or Mohammad Ghidary (03)

Please use a #2 pencil (provided).

Final exam info

Saturday, May 19 at 7:30 PM.
Review sessions:

```
mohammad: 1PM Wed 16 May
    dave: 1PM Thu 17 May
   jimin: 1PM Fri 18 May
```

- Prepare and e-mail questions in advance.
- All questions answered.
- No questions? No session.
- Any student may attend any or all sessions.