04- Backtracking

- recursive approach
- refinement of brute force method.
- limits search by eliminating candidates → don't satisfy certain constraints
- set of choice & don't know which leds to correct choice.
- each choice → partial soln → soln → DFS order.
- partial soln → not satisfied → on constraints → no explore further
- processing → state space tree.

Constraints

Explicit

- restrict each element x_i has to be chosen from given set only.
- depend on instant I of problem
- all tuples soln → satisfy explicit constraints

Implicit

- rules that decide which tuples in soln space of I satisfy criteria fn
- x_i → soln set → related to each other.

Appln

- 8 Queen's Problem
- Sub of subset problem
- Graph Coloring Problem
- Finding Hamiltonian cycle
- Knapsack

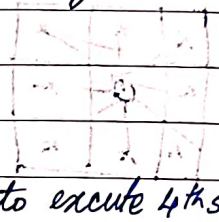
DOMS | Page No.
Date / /

* Control Abstraction

↳ flow of how it solves given problem.
↳ n-tuple.

$x = (x_1, x_2, \ldots, x_n) \rightarrow x_i = 0$ or $x_i = 1$.

$x_i \rightarrow$ choosen from a finite set of components s_i.
$x_i \rightarrow$ selected then set is as 1 or else as 0.
Backtracking tries to find a vector x that maximizes or minimizes criterion fucⁿ $p(x_1, x_2, x_3, \ldots x_n)$.



Complexity → $O(P(n)\, n!)$
- $P(n)$ → Time to execute first 3 steps can be computed in polynomial time.
- $n!$ → Time to excute 4th step

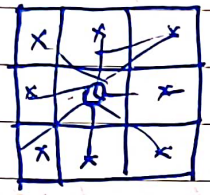
* Time Analysis

Four parameters →
1) Time to compute tuple $x[k]$
2) no. of $a[k]$ elements which satisfy explicit constraints
3) time required by bounding fuction Bk to generate feasible sequence.
4) no. of elements $a[k]$ → satisfy → bounding fuct Bk → values of k.

8 Queen Problem -

- given 8×8 chess board, arrange 8 queens in such a way that no two queens attack each other

Two queens attack → same row, column or diagonal



Queens Attack

8 queens problem → 92 solutions out of 12 soln that are fundament sol^n

80 soln → rotations or reflecting 12 sol^n

- special case of n queen problem
- N-Queens → N=3

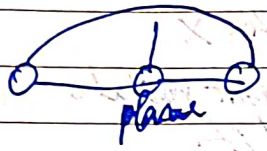
continue placing queens with row by row or col by col & check if bounding fun" is met.

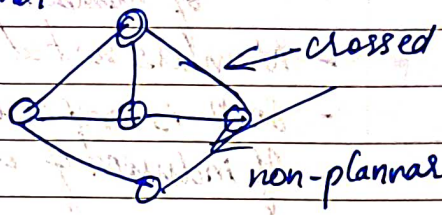
- If met → we continue exploring otherwise cutoff

DOMS | Page No.
Date / /

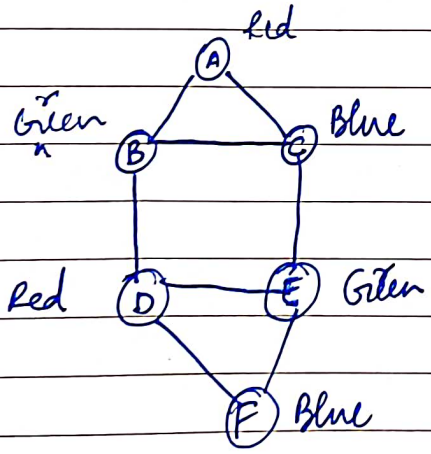
* Graph Coloring Problem -

graph → planar → drawn on 2D plane with no two edges crossing each other.





- for only planar graphs
- coloring vertices → graph → • no two graphs have same color.
  - no adjacent vertice has same color.
  - yet m- colors used.
- m- coloring problem
- least no of colors → color graph → chromatic numbes.
- Is a decision as well as optimization problem.



Complexity → no. of nodes increas exponentially at every level in state space tree

with M colors & n vertices

$$T(N) = 1 + M + M^2 + \dots + M^n$$

$$= \frac{M^{n+1} - 1}{M - 1}$$

$$T(N) = O(M^n)$$

Sum of Subset.

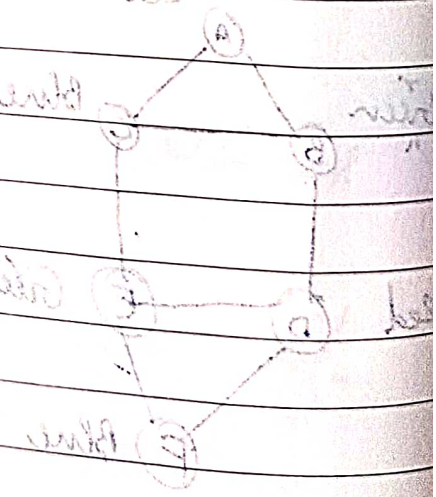
given a set of +ve integers find combination of numbers that sum to give a value N.

- Numbers sorted → ascending order.
- Soln is represented using vector X.
- If inclusion of one element → doesn't violate constraints add it.

Time complexity -

At level i we have 2^i nodes in statespace tree.

$$T(n) = 1 + 2 + 2^2 + \dots + 2^n = 2^{n+1} - 1 = O(2^n)$$



Branch & Bound

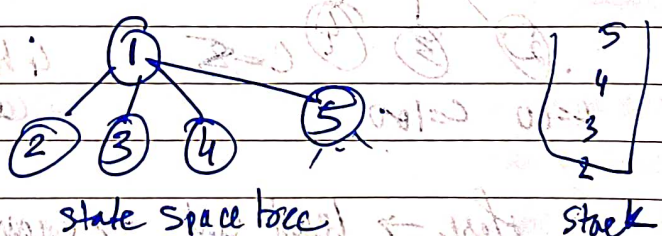
→ Backtracking used for decision problems.
→ optimization problems.
→ build state space tree → optimal solution. by pruning branches of tree which don't satisfy bound.
→ slower than dynamic programming.
→ where greedy or dynamic programming fail.
→ e-nodes are put in queue is called FIFO branch & bound where if put in stack it is called LIFO branch & bound.
→ heuristic maximize probability of better search or min. prob. of worst case.

Time analysis

→ BnB → combinatorial problems.
→ exponentially increasing time complexity.

Node to be explored is chosen in many ways.

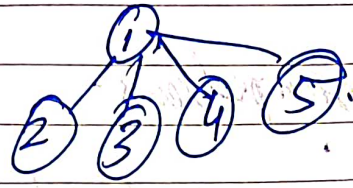
→ 1) FIFO



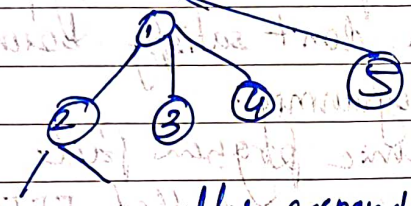
state space tree Stack

S → topmost node we will explore first.
top of stack → e node.

2) LIFO -



[2 3 4 5]
queue.

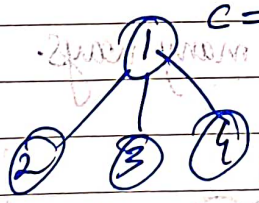


[3, 4, 5]
queue.

after expanding 2 its child nodes will go at basic of queue & 3 will expand.

3). Least Cost Branch & Bound (LC) -

cost fun^{n} → calculate heuristics of finding an optimal & prune out unoptimal or unnecessary branch or paths.



4 has least cost so, we expand it.

Here → least cost → expand node w/o stack/queue.

DOMS | Page No.
Date / /

Knapsack using Branch & Bound-

- ~~Calculate heuristic~~
- work minimization problem where D// Knapsack is a maxi. problem.
- To counter → choose negative upper bound & costs.
- upper bound → min cost that node can have.
  if livenode exceed upperbound → kill node.
  ↳ don't explore anymore.

Upper bound → $U = \sum_{i=1}^{k} p_i x_i$ Cost func^n → $\sum_{i=1}^{n} p_i x_i$ (with fraction)

Backtracking	Branch & Bound
i) soln → traced using DFS	i) soln → may not be DFS may use BFS.
ii) decision problem solved	ii) optimization problems
iii) bad choices possible.	iii) No bad choices
iv) state space tree is searched till soln obtained.	iv) all tree searched as optimum soln can be anywhere
v) Appln → M-coloring, Sequens	v) Appln – Job Scheduling, TSP.