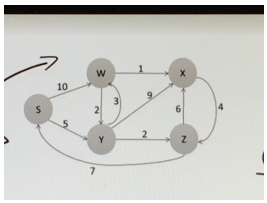
Dijkstra's algorithm

Choose node	
s(-, 0)	y(s,5), w(s,10), x(-,∞), z(-,∞)
y(s, 5)	z(y,7), w(y,8), x(y,14)
z(y,7)	w(y,8), x(z,13)
w(y,8)	x(w,9)
x(w,9)	



x - w - y - s — cheapest to x from s

w - y - s — cheapest to w from s.

z - y - s — cheapest to z from s.

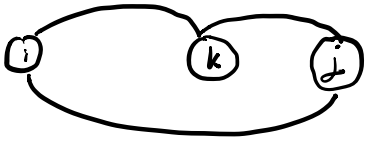
y - s — cheapest to y from s.

Warshall's algorithm

Dynamic solution of overlapping solutions

- A = Adjacency matrix
  0, 1 if 2 nodes (connected edges)
- Is some arbitrary node i reachable from some other node j?
- Transitive closure $A \to B \wedge B \to C$; $A \to C$

Label nodes 1 ... n



$R^0_{i,j}$ R^1, R^2 ... $R^4_{i,j}$

Direct links (k)

Recurrence:

R^k

$$R^k(i,j) = R^{k-1}_{i,j} \text{ or } \left(r^{k-1}_{i,k} \wedge r^{k-1}_{k,j}\right)$$

→ k goes from 1 → n n passes.

$n'(i,j) = n^0(i,j)$ or $n^0(i,k) \wedge n^0(k,j)$
↑ each iteration of k

n^0 (m = 4)

	1	2	3	4
1	0	1	0	0
2	0	0	0	1
3	0	0	0	0
4	1	∞	1	0

Adjacency matrix.

k = 1

i = 1 j = ~~1~~ 2 3 4

i = 2 j = ~~1~~ ~~2~~ ~~3~~ 4

i = 3 j = 1 2 3 4

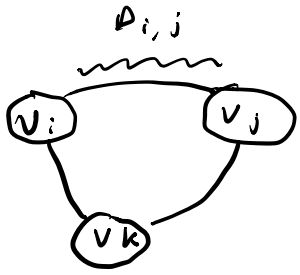
i = 4 j = 1

Floyd's walkthrough – all pairs shortest path.

→ Adjacency matrix

$D =$

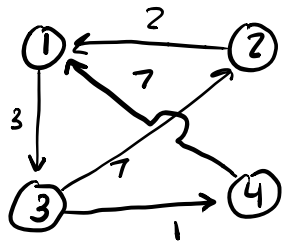
	i	j
n i	0	14
		0
		0
	∞	0



Build each iteration

k = 1 .. n . . . .

$D_{i,k} + D_{k,j}$ k as intermediate.
find least cost.



$$D^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{bmatrix} \end{array} \quad \begin{array}{l} i=1 \; j=1 \\ i=2 \\ i=3 \\ i=4 \end{array}$$

$$D(i,j) = D^0(i,j), D^0(i,1) + D^0(1,j)$$

$$D^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & 5 & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix} \end{array} \quad \begin{array}{l} i=1 \; j=1 \\ i=2 \\ i=3 \\ i=4 \end{array}$$

$k = 1 \ldots n$

$$D^k(i,j) = \min \{ D^{k-1}(i,j), D^{k-1}(i,k) + D(k,j) \}$$

Backtracking

- Planning, solutions w/ very large solution space.
- Problems that are impossible to solve.
- Combinatorial / exponential
- exhaustive search
- eliminates useless / impossible subsets of results.

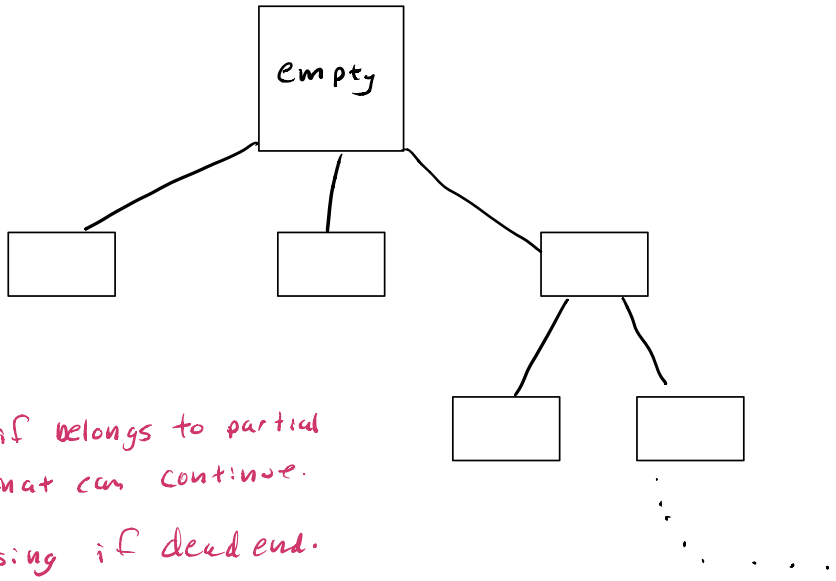
1) Construct a potential solution 1 component at a time.

search space. 2) If there are no potential values of remaining components then we do not continue w/ this potential solution.

-> search space tree: D-F

Back tracking space tree generation.

- construct by adding a component to existing state as long as adding compone up will not violate a constraint
- If no options for next component end state path
- Backtrack up a level to prior state and try next component.

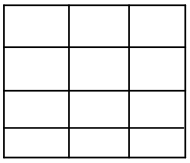


- Promising if belongs to partial solution that can continue.
- Non promising if dead end.

n-queens problem

4 queens

NxN chess board



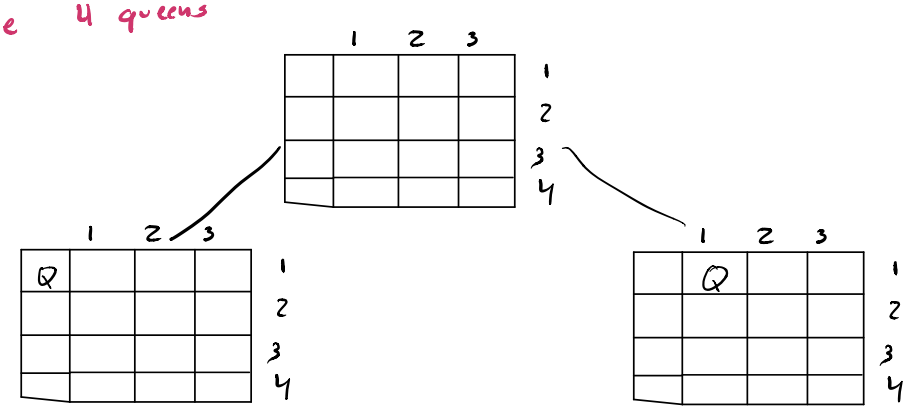
Place queens on board so that no queen can attack another

n=1 trivial

n=2 no

n=3 no

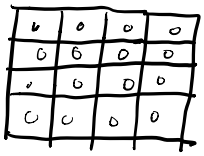
Trace 4 queens



General Backtracking logic

Partial sol'n first : components

```
Algorithm Backtrack (x[1...i])
  if x[1...i] A solution returns it
  else
      for each element ∈ S_{i+1} [within constraints]
          x[i+1] ← x
          Backtrack (x[1...i+1])
```

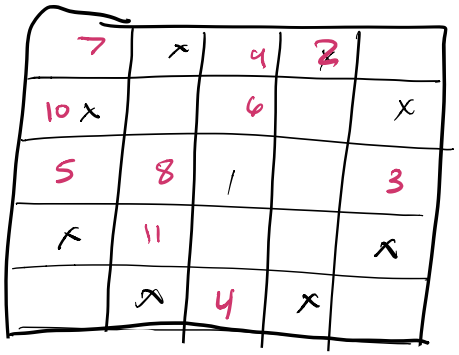


board

Check safe (Board, row, col)

Board [row, col] is it safe? If yes, place queen on board at the row and column.

```
bool placeQueens( board, N, row) {
    if (n==2 || n==3) return false
    if ( row == N) return true // Board complete
  for(int col= 0; col<N; col++) {
        if (placement Safe ( board, N, row, col))
            board[row][col]=1;
            if (place queens (board, N, row+1))
                return true ;
            else
                board[row][col] = 0;
    }
    return false;
}
```



knight visits every square.

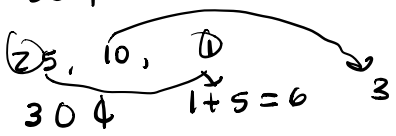
loop { move 1, move 2, move 3, move 4 } 8 possible moves

↳ Greed technique.

Making Change

25, 10, 5, 1

30 ¢ → 25 + 5, 2 coins.



Optimal sol'n
→ least # of coins possible.

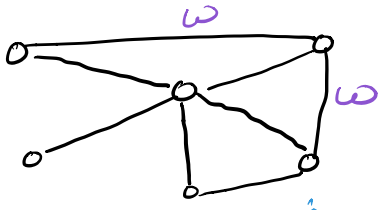
Greedy approach → take best available alternative at each step and hope that a sequence of "best" alternatives leads to best optimal sol'n.

1) Feasible constraints
2) locally optimal. - all feasible
3) irrevocable.

9.1 Prim's algorithm

-> minimum spanning tree problem.

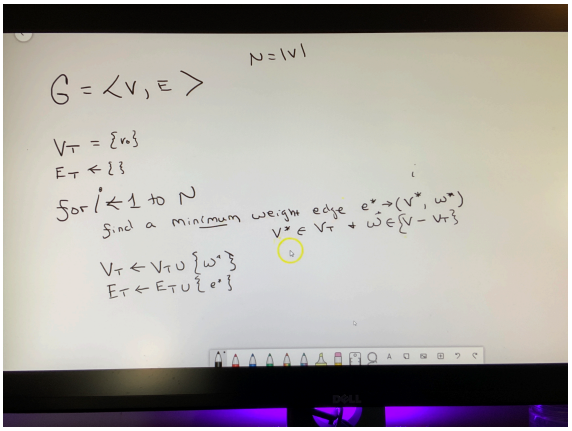
Subgraph connected.
no cycles (tree).
min total weight (cost)

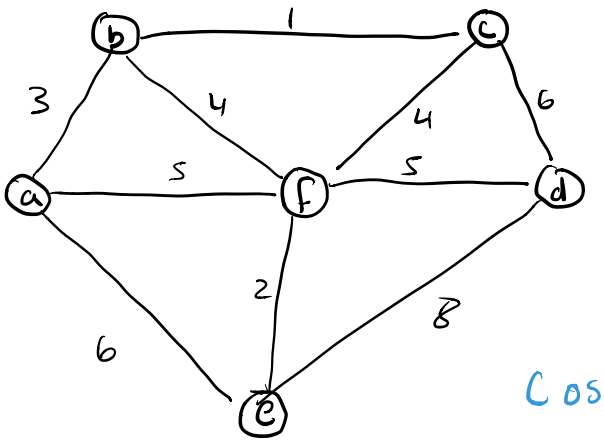


Exhaustive -> generate all spanning tree & pick cheapest.

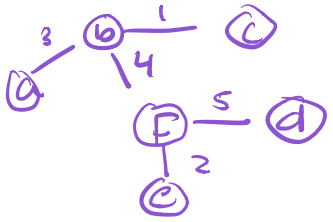
Prim's

- Iteratively construct minimal spanning tree by adding one vertex at a time.
- Choose next vertex not in tree w/ cheapest edge to tree.





Prim's trace



Cost = 1+3+4+5+2 = 15.

Choose a to add to set.

b(3, a), c(∞, -), d(∞, -), e(6, ∞), f(5, a)

Choose b (cheapest)

c(b, 1), d(∞, -), e(6, a) f(4, b)

Choose c

d(6, c) e(6, a) f(4, b)

Choose f

d(f, s) e(f, 2)

Choose e

d(s, f) → s < 8 !

Complexity

each pass -> choose node. $|V|$ $N \cdot N-1$

↳ { create list of edges reachable from node. } $N-1$ $|V|^2$

Complete

$N-1$

---

kruskals algorithm

↳ # of edges.

↳ minimum spanning tree 1 element at a time by choosing edges that do not create cycles.

kruskals

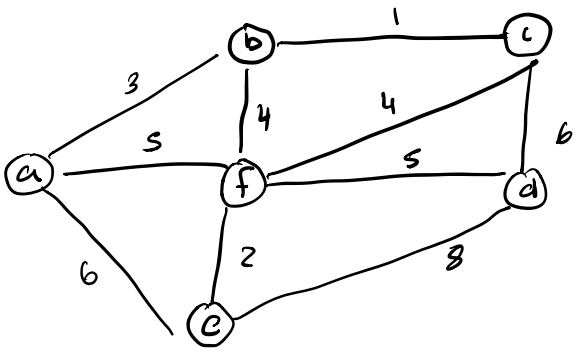
$G \leftarrow \langle V, E \rangle$

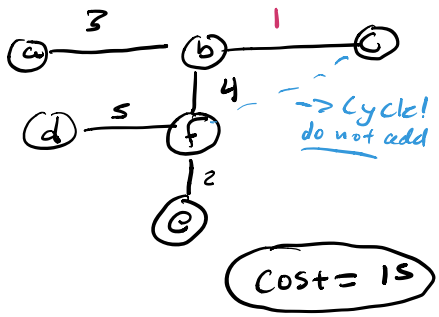
$E_T \leftarrow \{\}$

sorted edges by weight

iterate through node count.

choose cheapest edge that connects components w/ creating a new cyclic.





Cost = 15

Sort edges

~~bc~~ 1 | ef 2 | ~~ab~~ 3 | bf 4 | ~~df~~ 4 | cf 4 | af 5 | df 5 | ae 6 | cd 6 | de 8

cf: cycle, df: cycle

$n \log n$