2.8.2017

General Framework for a Backtracking Algorithm

- assume that the solution has the form

$$x[1], x[2], \ldots, x[n]$$

where the values $x[i] \in S$

Note: in the previous example $S = \{1, 2, \ldots, n\}$

```
backtrack(n)
rbacktrack(1,n)

rbacktrack(k,n)
for each x[k] ∈ S
    if bound(k) == true
        if k == n
            output a solution; stop here if only one solution is desired
        else // k<n
            rbacktrack(k+1,n)

bound(k)
// give pseudocode implementation
```

$$\underbrace{x[1], \ldots, x[k]}\ \overrightarrow{\quad\quad}\ \ldots\ x[n]$$

- function bound(k)
  - assumes that $x[1], x[2] \ldots x[k-1]$ is a partial feasible solution and that $x[k]$ has been assigned some value
  - returns
    - true if $x[1], x[2], \ldots, x[k]$ is a partial feasible solution
    - false otherwise
- goal: design an efficient bound() function that eliminates many potential nodes from the search tree

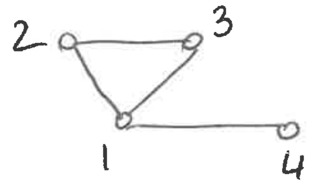
Graph representation (review)

- adjacency-matrix representation
- adjacency-list representation

Adjacency-matrix representation of a graph $G=(V,E)$

- use a $|V| \times |V|$ matrix $A=(a_{ij})$

$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

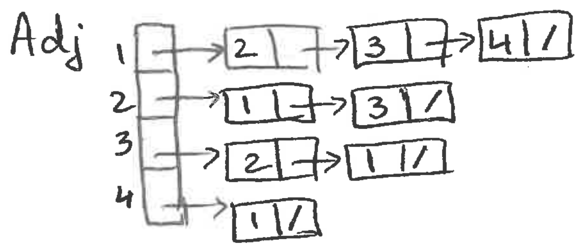


$$A \quad \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 1 & 1 \\ 2 & 1 & 0 & 1 & 0 \\ 3 & 1 & 1 & 0 & 0 \\ 4 & 1 & 0 & 0 & 0 \end{array}$$

- space: $\Theta(V^2)$
- RT to find whether $(u,v) \in E$ is $\Theta(1)$

Adjacency-list representation of a graph $G=(V,E)$

- use an array of linked-lists with one linked-list for each vertex



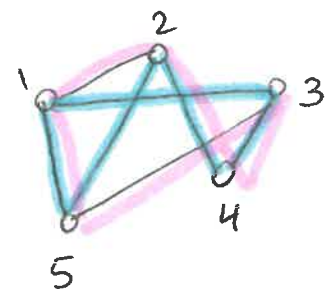
- space $\Theta(V+E)$
- RT to find whether $(u,v) \in E$ is $O(u.\text{degree})$

- if the graph G is
  - <u>sparse</u>, then the adjacency-list representation is preferred
  - <u>dense</u>, then the adjacency-matrix representation is preferred

The Hamiltonian-Cycle (HC) Problem

Problem definition: given a graph $G=(V,E)$ undirected, find whether G has a HC (a cycle that contains each vertex exactly once).

example



G has a HC

$HC = (1,3,4,2,5)$

- HC problem is NP-complete
- we can represent the solution using an array

x

1	2	3	4	5
1	3	4	2	5

so the solution has the form $x[1], x[2], x[3], \ldots, x[n]$

- we can assume without loss of generality that $x[1]=1$
- the alg returns
  - true if G has a HC ; stop as soon as the alg. finds a HC
  - false if G has no HC
- assume that G is represented using the adjacency-matrix "adj"
- our alg. follows the general framework for a backtracking alg. and uses the functions:
  - hamilton(adj, x)
  - rhamilton(adj, k, x)
  - path_ok(adj, k, x)

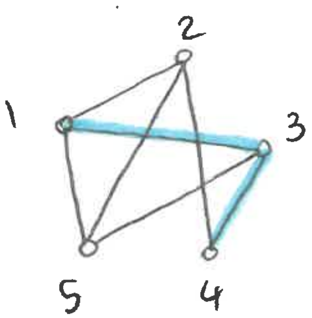
rhamilton(adj, k, x)

- tries to select the k^{th} vertex in the HC
- assumes that $x[1], x[2], .., x[k-1]$ forms a partial feasible solution

path_OK(adj, k, x)

- assumes that $x[1], .., x[k-1]$ is a partial feasible solution and that $x[k]$ has been assign some value
- returns
  - true if $x[1], ..., x[k]$ is a partial feasible sol.
  - false otherwise

- When is $x[1], .., x[k]$ a <u>feasible</u> partial sol?



check if
- $x[k]$ is different than $x[1], x[2], .., x[k-1]$
- $k<n$, check if $(x[k-1], x[k])$ forms an edge
  $k=n$, check if $(x[n-1], x[n])$ and $(x[n], x[1])$ are edges

<u>$k=4$</u>

	1	2	3	4	5
x	1	3	4	?	

- How can we remember which vertices are already used?

array used[1..n]

used[v] = true if v has been already included in the path
false, otherwise

<u>example</u>:

$k=4$

	1	2	3	4	5
x	1	3	4	?	

	1	2	3	4	5
used	T	F	T	T	F

input param → adj, output param → x

```
hamilton(adj, x)
  n = adj.last    // n is the number of vertices
  x[1] = 1
  used[1] = true
  for i = 2 to n
      used[i] = false
  rhamilton(adj, 2, x)

rhamilton(adj, k, x)
  n = adj.last    // n is the number of vertices
  for x[k] = 2 to n
      if path_OK(adj, k, x) == true
        used[x[k]] = true
        if k == n
            print solution x[1], x[2], .., x[n]
            return TRUE
        else // k<n
            if rhamilton(adj, k+1, x) == true
               return true
        used[x[k]] = false
  return false

path_OK(adj, k, x)
  n = adj.last   // n is the number of vertices
  if used[x[k]] == true
     return false
  if k < n
     return adj[x[k-1], x[k]]
  else // k=n
     return adj[x[n-1], x[n]] && adj[x[n], x[1]]
```

$\underbrace{x[1] \ldots x[k]}$ $\ldots x[n]$

RT analysis

- How many times is rhamilton called?

$k=1$ 0 times

$k=2$ 1 time

$\underline{1}\ \underline{x}\ \underline{?}$ $k=3$ $\le (n-1)$ time

$\underline{1}\ \underline{x}\ \underline{x}\ \underline{?}$ $k=4$ $\le (n-1)(n-2)$ times

$\vdots$

$k=n$ $\le (n-1)(n-2)\cdots 2$ times

- rhamilton takes $\Theta(n)$ besides the recursive calls

$$RT \le n\left(1+(n-1)+(n-1)(n-2)+\ldots+(n-1)(n-2)\cdots 2\right)$$

$$RT \le n\cdot(n-1)!\,\Big(\underbrace{\frac{1}{(n-1)!}+\frac{1}{(n-2)!}+\frac{1}{(n-3)!}+\ldots+\frac{1}{1!}}_{=e-1}\Big)$$

$$\boxed{\sum_{i=0}^{\infty}\frac{1}{i!}=e}$$

$$RT \le n!\,(e-1)$$

$$\boxed{RT = O(n!)}$$