Introduction to combinatorial optimization, modeling and complexity theory

Part I : Introduction to combinatorial optimization and graph theory

- Beginnings of Operations Research;
- Graph theory: basic notions
  - Connectivity, shortest path problems, algorithms, applications in routing in Internet,
- Modeling combinatorial problems through LP, examples
- Integer linear programming
- exercises;

Part II : Introduction to Computational Complexity Theory

- Algorithmic complexity
  - Notions and evaluation measures, examples
- Problems complexity
  - Decision problems, P and NP classes, polynomial reduction;
  - NP-completeness, Cook's Theorem, relation P vs NP, examples, exercises;
  - Pseudo-polynomiality, dynamic programming, NP-complete problems in the strong sense, examples, exercises.

History…

- **Léonard EULER**: 1707-1783
  - Seven Bridges of Königsberg

- **Charles BABBAGE**: 1791-1871(**Ada LOVELACE** : 1815-1852**)**
  - **Design of computers:** Babbage sought a method by which mathematical tables could be calculated mechanically, removing the high rate of human error.

- **Alan TURING:** 1912-1954
  - The Turing machine
  - Decrypting the Enigma code (**Combinatorial**),
  - COLOSSUS: one of the first computers

History…

- **Léonid KANTOROVITCH : 1912- 1986**
  - a pioneer of linear programming… transport program,
  - Nobel price in economics (1975)
- **Georges Bernard DANTZIG : 1914- 2005**
  - Linear programming
- **Jeff HAWKINS : 1957**
  - Inventor of personnel-assistant (Palm Pilot)

History…

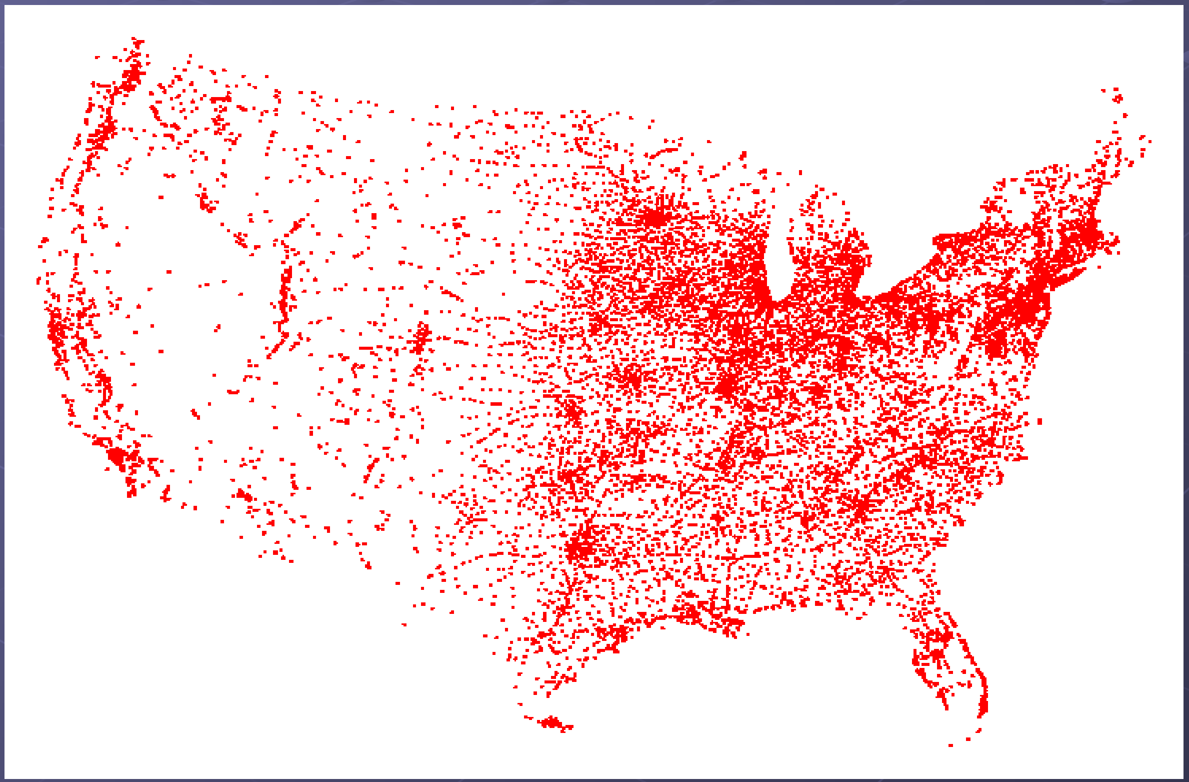
- **Paul ERDOS and Alfred RENYI**
- **Albert-Lazlo BARABASI, Claude BERGE, Ken APPEL and Wolfgang HAKEN,**
- **Jack EDMONS, Bernard Roy, Paul ROBERTSON et Neil SEYMOUR, Robert TARJAN**

Some problems of Operations Research

- *Discrete combinatorial problems*
  - *Travelling salesman problem,*
  - *Minimum spanning tree*
- *Continuous combinatorial problems*
  - *Linear programming,*
- *Random problems*
  - *Queuing theory*
  - *Equipment replacement*
- *Competitive situations*
  - Game theory

Discrete combinatorial problems Travelling salesman problem

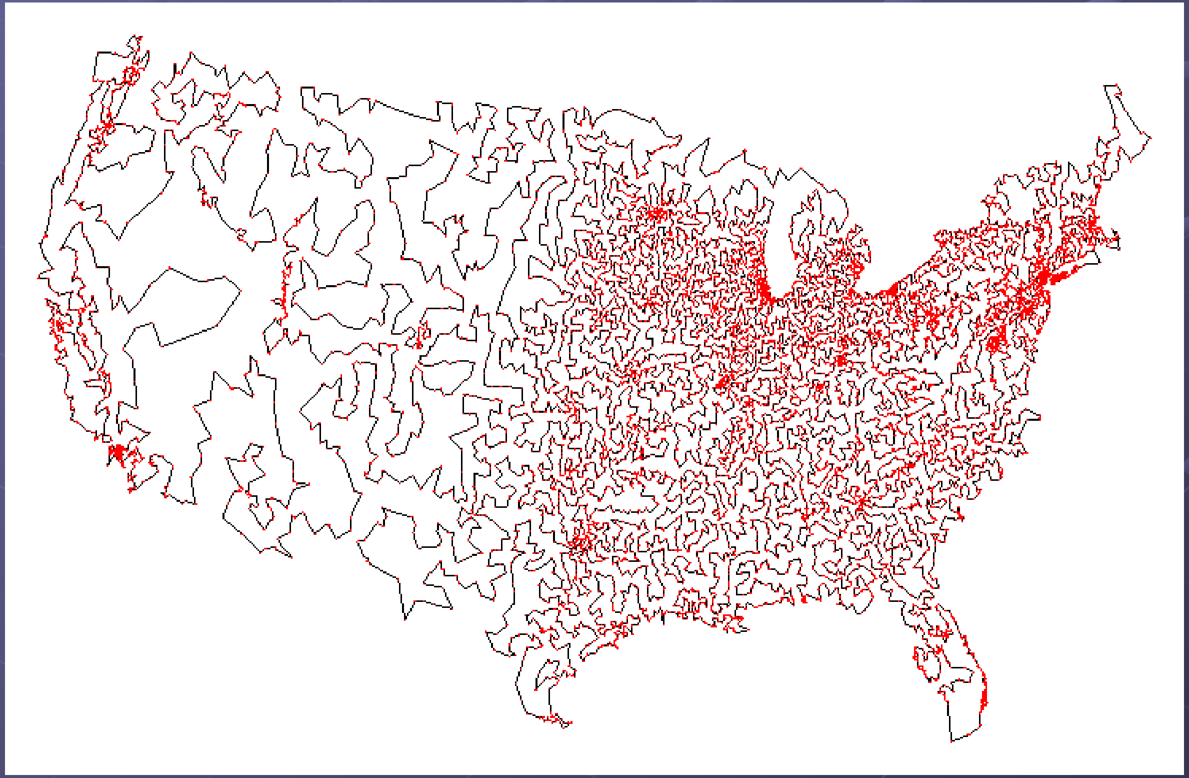
- TSP. Given a set of n cities and a pairwise distance function d(u, v), is there a tour of length ≤ D?



All 13,509 cities in US with a population of at least 500
Reference: http://www.tsp.gatech.edu

Discrete combinatorial problems Travelling salesman problem

- TSP. Given a set of n cities and a pairwise distance function d(u, v), is there a tour of length $\leq$ D?



Optimal TSP tour
Reference: http://www.tsp.gatech.edu

Continuous combinatorial problems Linear programming

Example. Suppose that a farmer has a piece of farm land, say *A* square kilometers large, to be planted with either wheat or cereals or some combination of the two.

The farmer has a limited permissible amount *F* of fertilizer and *P* of insecticide which can be used, each of which is required in different amounts per unit area for wheat (*F*1, *P*1) and cereals (*F*2, *P*2).

Let *S*1 be the selling price of wheat, and *S*2 the price of cereals. If we denote the area planted with wheat and cereals by *x*1 and *x*2 respectively, then the optimal number of square kilometers to plant with wheat vs cereals can be expressed as a linear programming problem:

maximize $S_1x_1 + S_2x_2$ (maximize the revenue)

subject to:

$x_1 + x_2 \leq A$ limit on total area)

$F_1x_1 + F_2x_2 \leq F$ (limit on fertilizer)

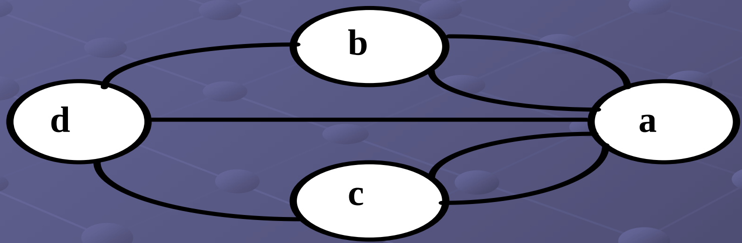
$P_1x_1 + P_2x_2 \leq P$ (limit on insecticide)

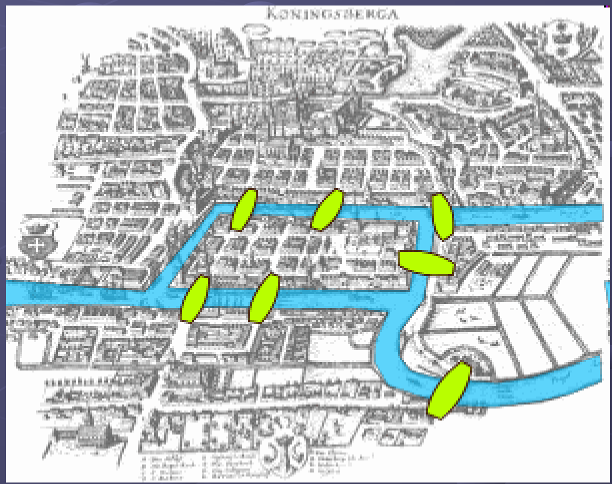
$x_1 \geq 0,\ x_2 \geq 0$ (cannot plant a negative area)

Random problems

- *Queuing theory*
  - *applications to (internet) network congestion;*
  - *ordering the take-off of aircraft.*
- *Equipment replacement*
  - *Deciding the replacement date for equipments with given failure probability.*

Why using graphs?





Seven bridges of Königsberg

Given the above graph, is it possible to construct a path (or a cycle, i.e. a path starting and ending on the same vertex) which visits each edge exactly once?

Basic definitions

- Directed Graphs
- Non-oriented graphs
- walks, cycles, paths, circuits
  - elementary
  - simple
  - Eulerian
  - Hamiltonian,
- Stable, coloring, clique…

Basic definitions

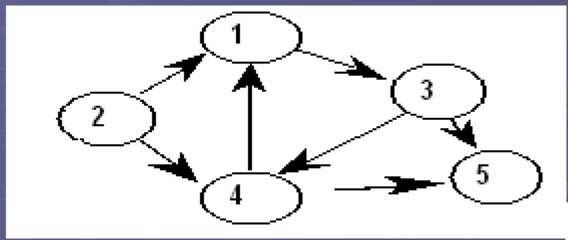
- A **directed graph** or **digraph** is a pair *G*= (*X*, *U*) of:
  - a set *X*, whose elements are called **vertices** or **nodes,**
  - a set *U* of ordered pairs of vertices, called **arcs**, **directed edges**, or **arrows**.
- It differs from an ordinary, or undirected graph in that the latter one is defined in terms of edges, which are unordered pairs of vertices.
- A valuated graph is G = (X, U, v) where (X, U) is a graph and v an application from U to R (real numbers).

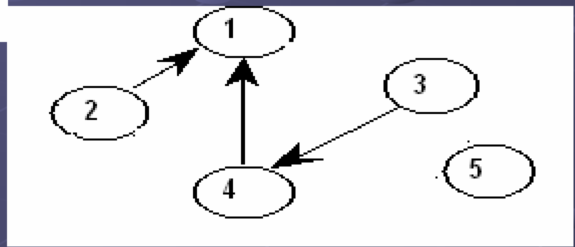
- Successors, predecessors, vertex degrees…

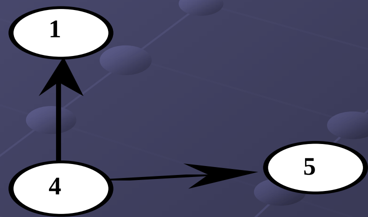
Basic definitions

- A **walk** is an alternating sequence of vertices and edges, beginning and ending with a vertex, where each vertex is incident to both the edge that precedes it and the edge that follows it in the sequence, and where the vertices that precede and follow an edge are the end vertices of that edge. A walk is **closed** if its first and last vertices are the same (called a **cycle)**, and **open** if they are different (called a path).
- The **length** *l* of a walk is the number of edges that it uses.
- A directed path is when edges are "has the same orientation"
- A directed cycle: without the arrows, it is just a cycle.
- A path is **simple (resp. elementary)**, meaning that no vertices (resp. no edges) are repeated.
- A graph is **acyclic** if it contains no cycles;
- A path or cycle is **Hamiltonian (resp. Eulerian)** if it uses all vertices (resp. edges) exactly once.

Associated graphs



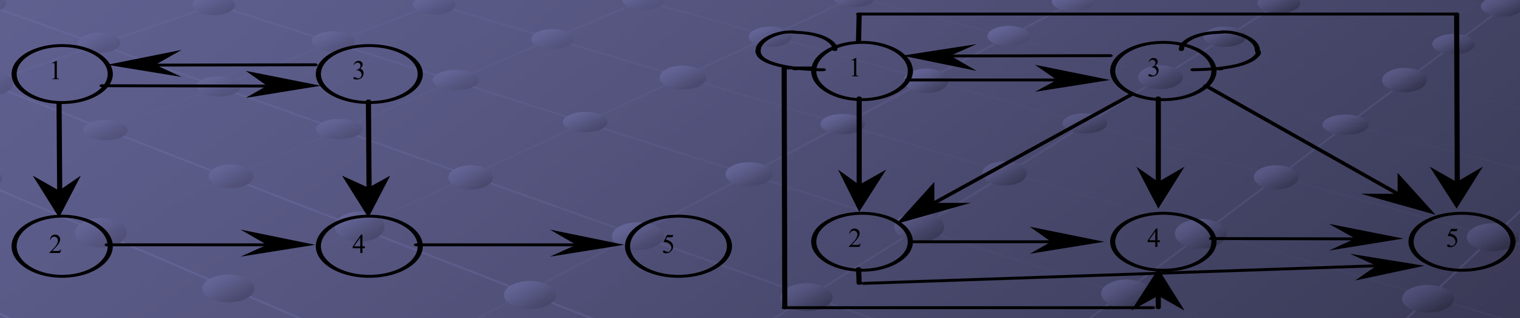




- Partial Graph
- Sub-graph
- Complementary graph

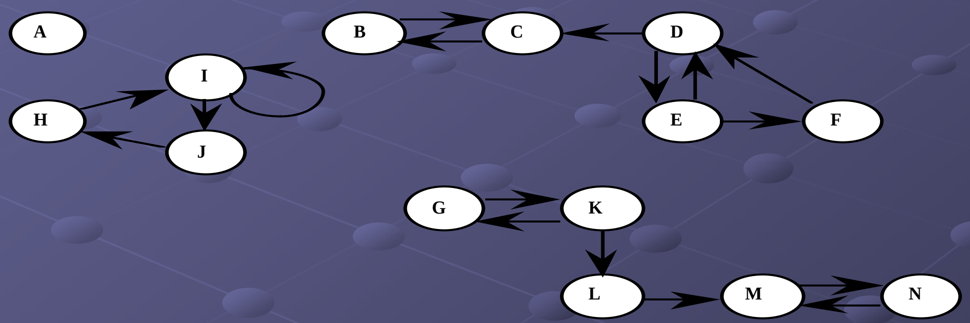
Associated graphs

- Transitive closure

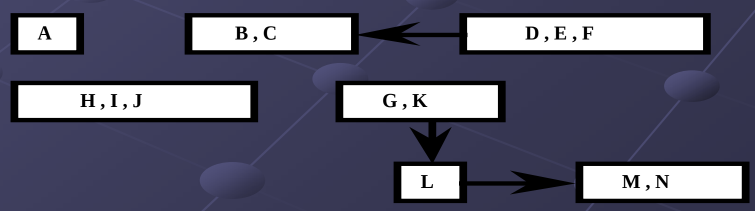


Connectivity

- Simple **Connectivity**: connected component
- Strong **Connectivity**: strong connected component



- Reduced Graph:



Connectivity and strong connectivity relations

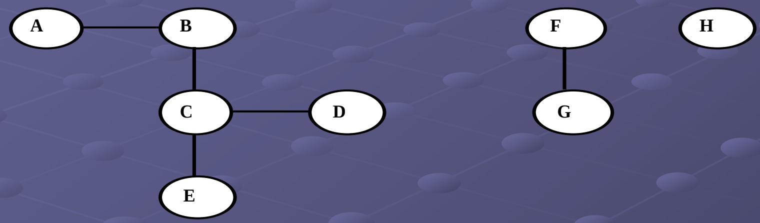
- An **equivalence relation** is a binary relation on a set that specifies how to split up (i.e. partition) the set into subsets such that every element of the larger set is in exactly one of the subsets.
  - *reflexive*, *symmetric* and *transitive*.

- **equivalence class of** *x* in *E* , denoted R(x), is given by: R(x)={y: xRy}.

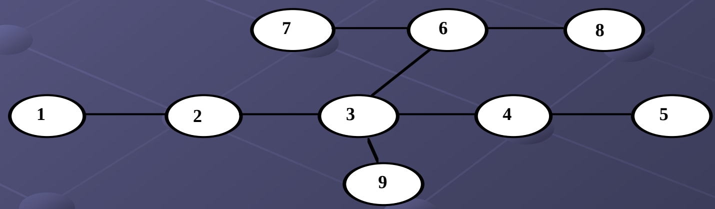
- *What can-we say about connectivity and strong connectivity relations?*

Particular Graphs

- Forest is a graph without cycle

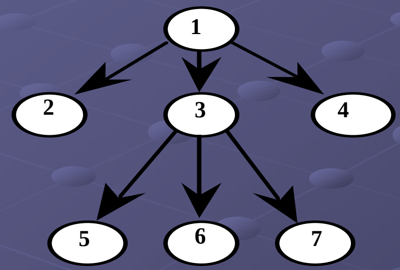


- Tree is a connected graph without cycle

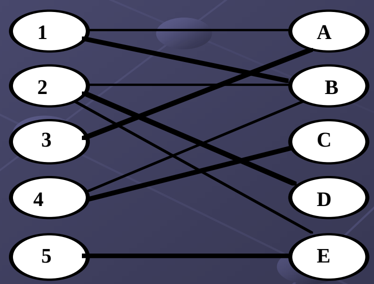


Particular Graphs

- In graph theory, an **arborescence** is a directed graph in which, for a vertex *v* called the root and any other vertex *u*, there is exactly one directed path from *v* to *u*.

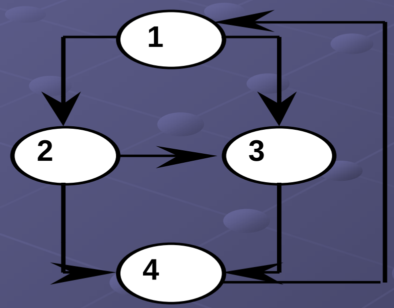


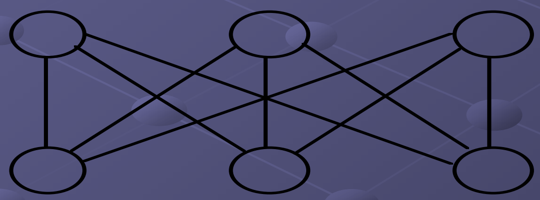
A bipartite graph is a graph whose vertices can be divided into two disjoint sets *U* and *V* such that every edge connects a vertex in *U* to one in *V*;



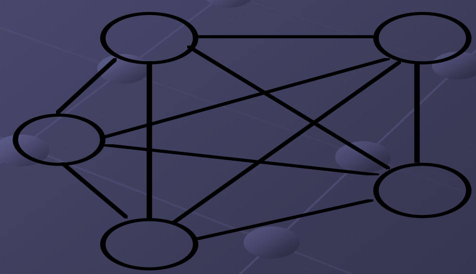
Particular Graphs

- In graph theory, a **planar graph** is a graph which can be embedded in the plane, i.e., it can be drawn on the plane in such a way that its edges intersect only at their endpoints.





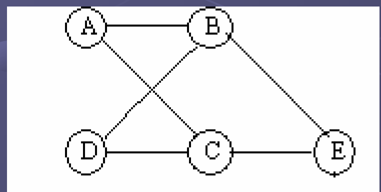
Graph of 3 entreprises ()

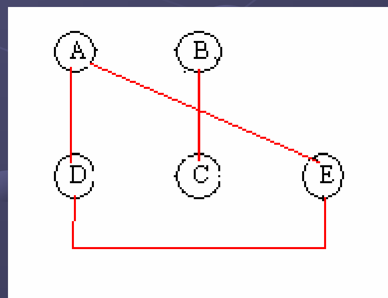


Complete Graph of 5 vertices ()

Basic definitions

- An **independent set** or **stable set** is a set of vertices in a graph no two of which are adjacent. The size of an independent set is the number of vertices it contains ($\alpha(G)$).
  - A maximal independent set is an independent set such that adding any other node to the set forces the set to contain an edge.
- A **clique** in an undirected graph G, is a set of vertices V, such that for every two vertices in V, there exists an edge connecting the two.

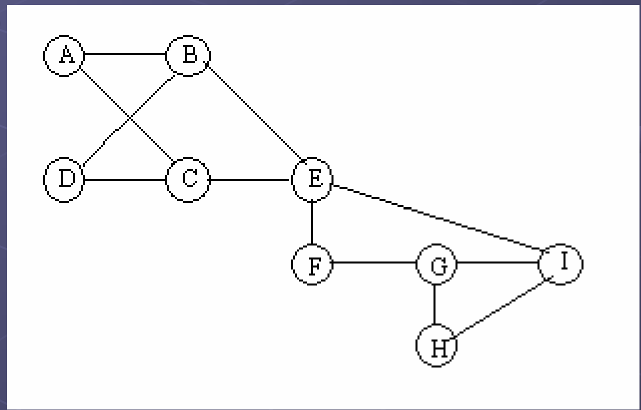




A **clique** in a graph G gives corresponds to a stable in its complementary graph and vice-versa.

Basic definitions

- Some graph G is called **c-chromatic** if its vertices can be colored with c colors such that no two adjacent vertices have the same color. Similarly, an **edge coloring** assigns a color to each edge so that no two adjacent edges share the same color.

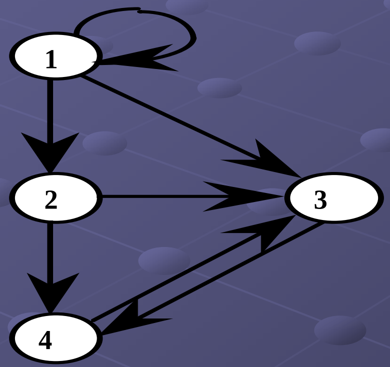


Conjecture of 4 colors (1875 Pertersen) : “all planar graphs are 4-chromatic”

Four colors theorem

Coding a graph

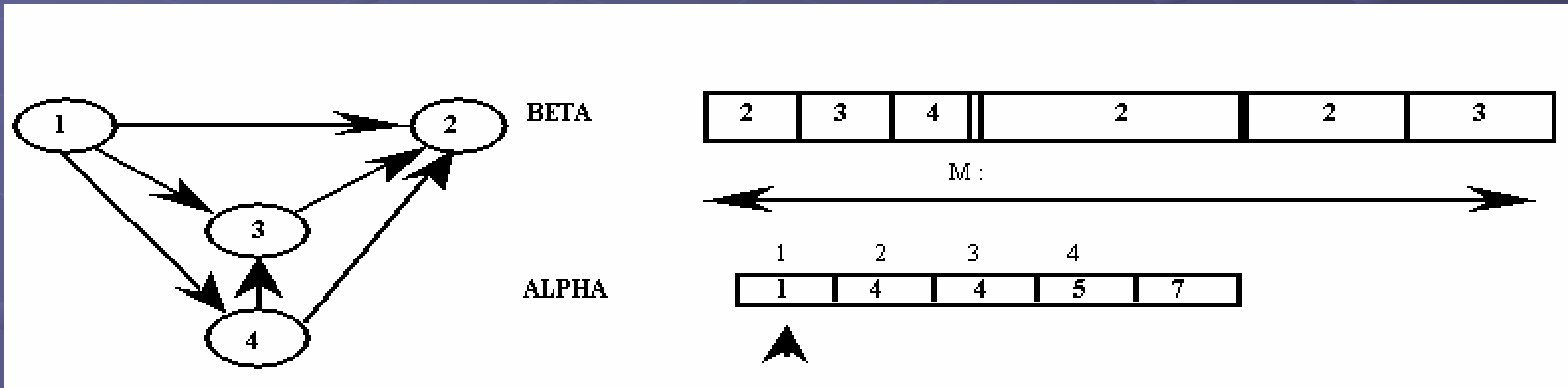
- Adjacency Matrix



$$\begin{vmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{vmatrix}$$

Coding a graph

- Successor queue β and α



Exercise: Write an algorithm which allows the passage from the successor queue to the adjacency matrix.

Shortest path problems

- Shortest path properties;
- Polynomial algorithms for shortest path computation, examples and complexity:
  - *Label correcting algorithms* : Ford algorithm;
  - *Label setting algorithms* : Dijkstra algorithm, Bellman algorithm;

Shortest path problems

- **Some properties:**
  - **Lemma 1**
    - Any path extracted from a shortest path is also the shortest one.
  - **Lemma 2**
    - A necessary and sufficient condition of existence of shortest paths is the absence of negative circuits.
  - **Lemma 3**
    - Let G be a graph without negative circuits and λ_i the shortest path values from x_0. A necessary and sufficient condition for that edge (x_i, x_j) is in a shortest path is : $\lambda_j - \lambda_i = v_{ij}$.

FORD Algorithm

Algorithm:

(i) *Initialization*

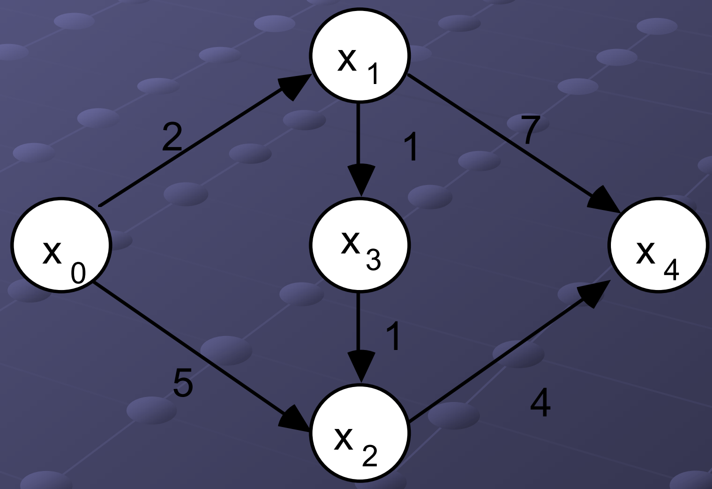
Poser $\lambda_0 = 0$ et $\lambda_i = +\infty$ pour $i > 0$.

(ii) *Edges examination*

for each vertex x_i, check all (x_i, x_j) from x_i and substitute λ_j with $\lambda_j + v_{ij}$ when $\lambda_i + v_{ij} < \lambda_j$.

(iii) Stop Test

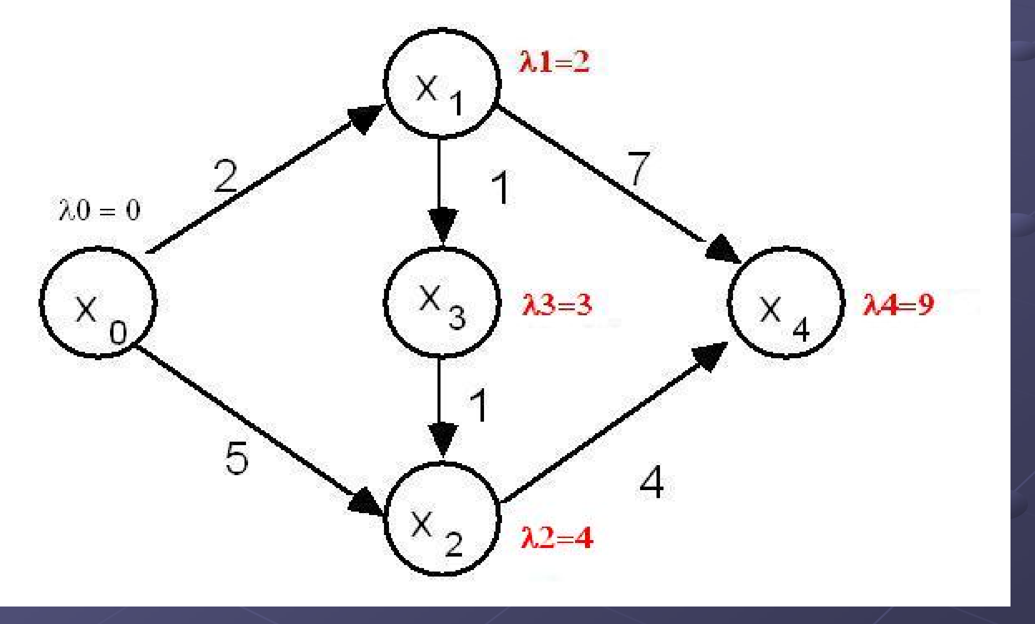
Iterate (ii) until some λ_j is updated in (ii).



An example

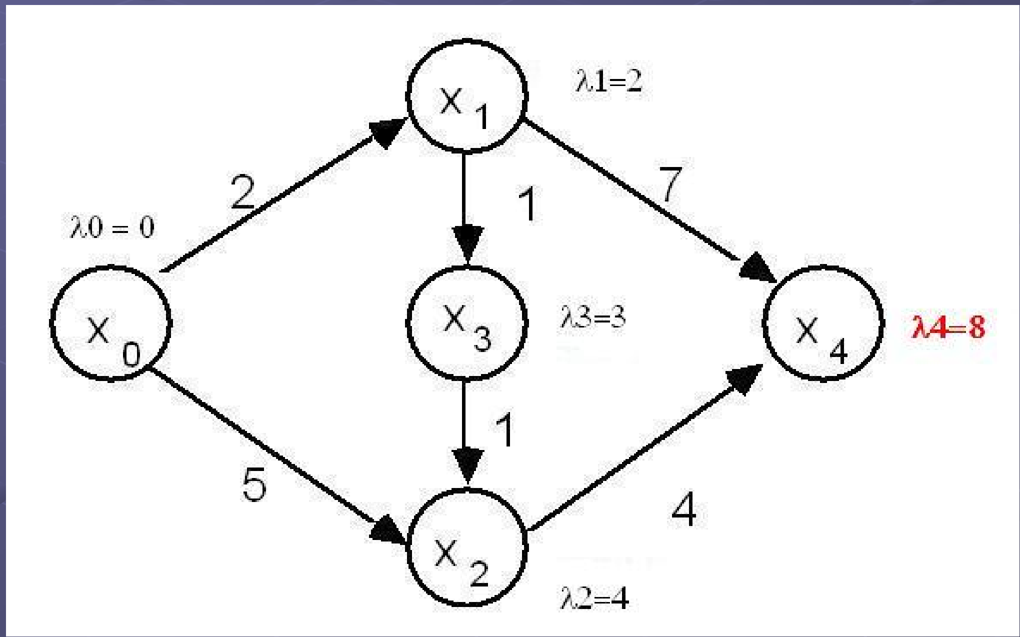
FORD Algorithm
an example

End of first iteration



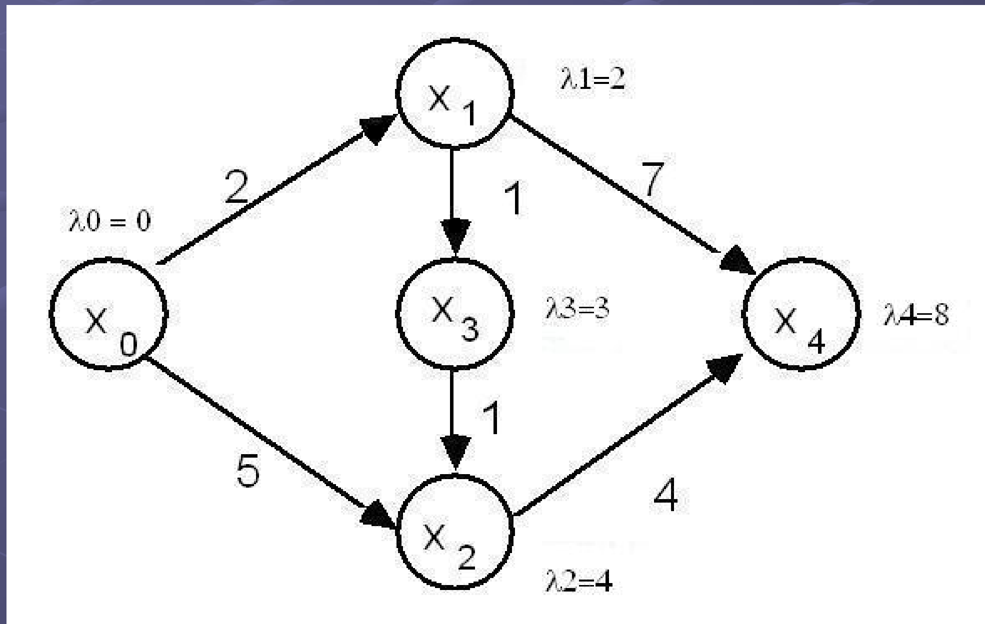
FORD Algorithm
an example

End of second iteration



FORD Algorithm
an example

Last iteration



Validity and complexity of Ford algorithm

Theorem 1:

Ford computes values of the shortest path from x_0 when the graph is without negative circuits.

Theorem 2:

The complexity of Ford algorithm is in $O(nm)$ where $n = |X|$ and $m = |U|$.

DIJKSTRA Algorithm

<u>Algorithm</u>

(i) set $S = \{x_0\}$, $\lambda_0 = 0$, $\lambda_i = v_{0i}$, if $(x_0, x_i) \in U$, and $\lambda_i = +\infty$, otherwise.

(ii) <u>while $S \neq X$ do:</u>

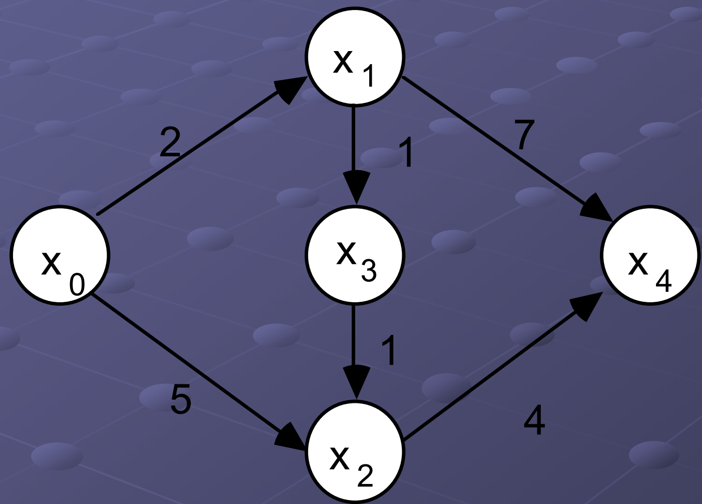
choose $x_i \in X - S$ of λ_i minimum.

set $S = S + \{x_i\}$.

For any $x_j \in (X - S)$, successor of x_i,

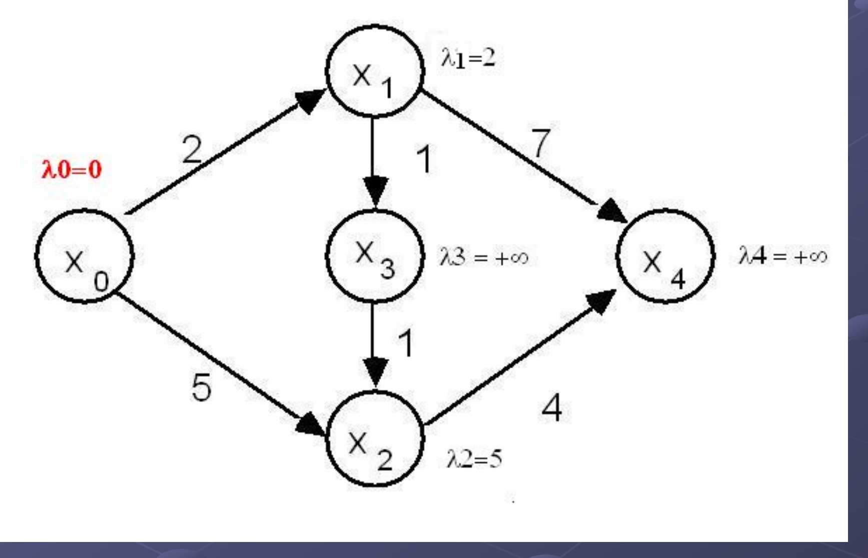
set: $\lambda_j = \min(\lambda_i + v_{ij}, \lambda_j)$.

DIJKSTRA Algorithm
an example



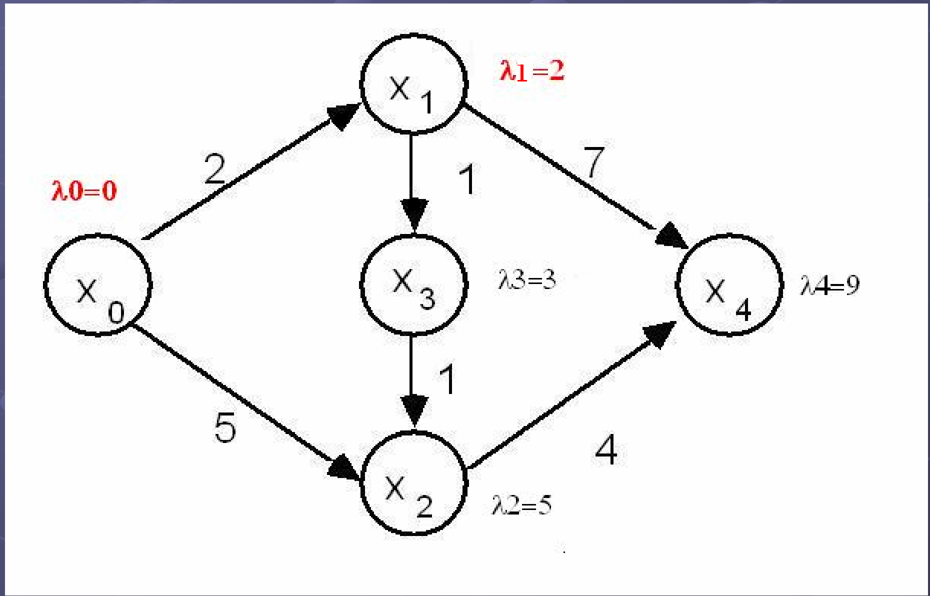
DIJKSTRA Algorithm an example

End of the first iteration



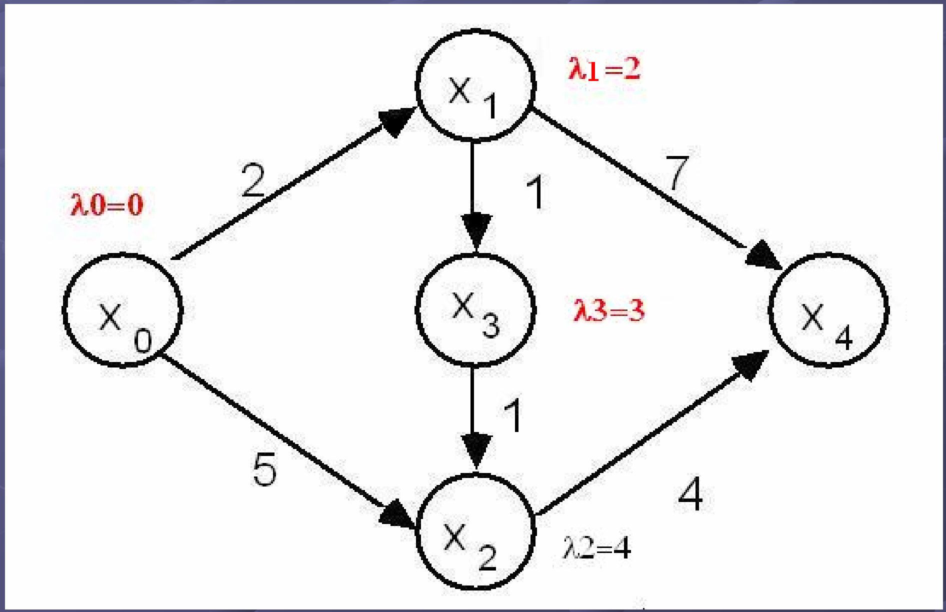
DIJKSTRA Algorithm an example

End of the second iteration



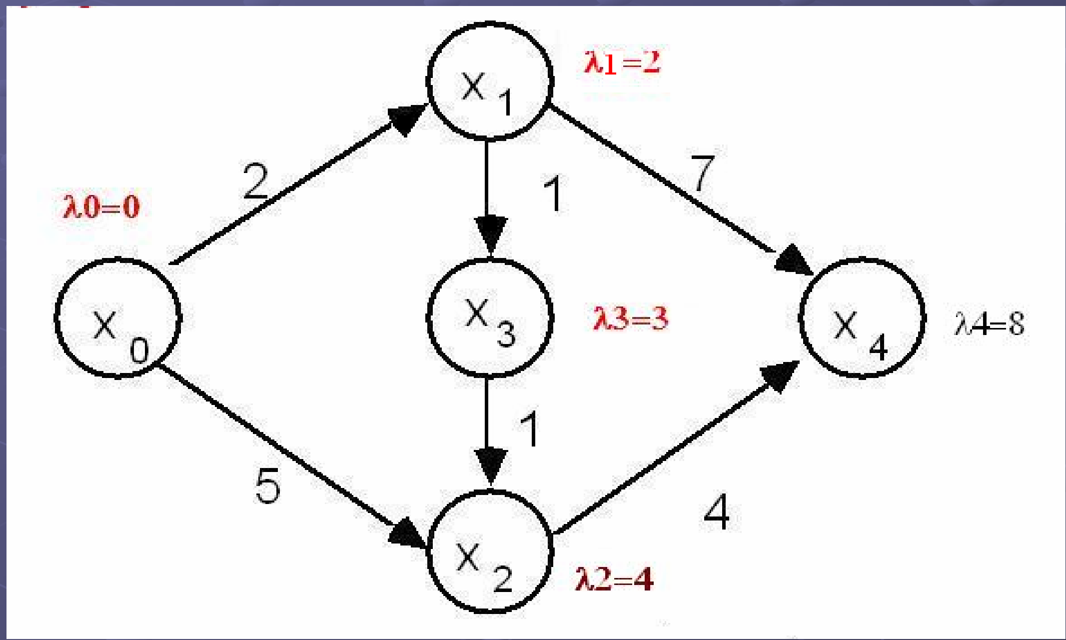
DIJKSTRA Algorithm
an example

End of the third iteration



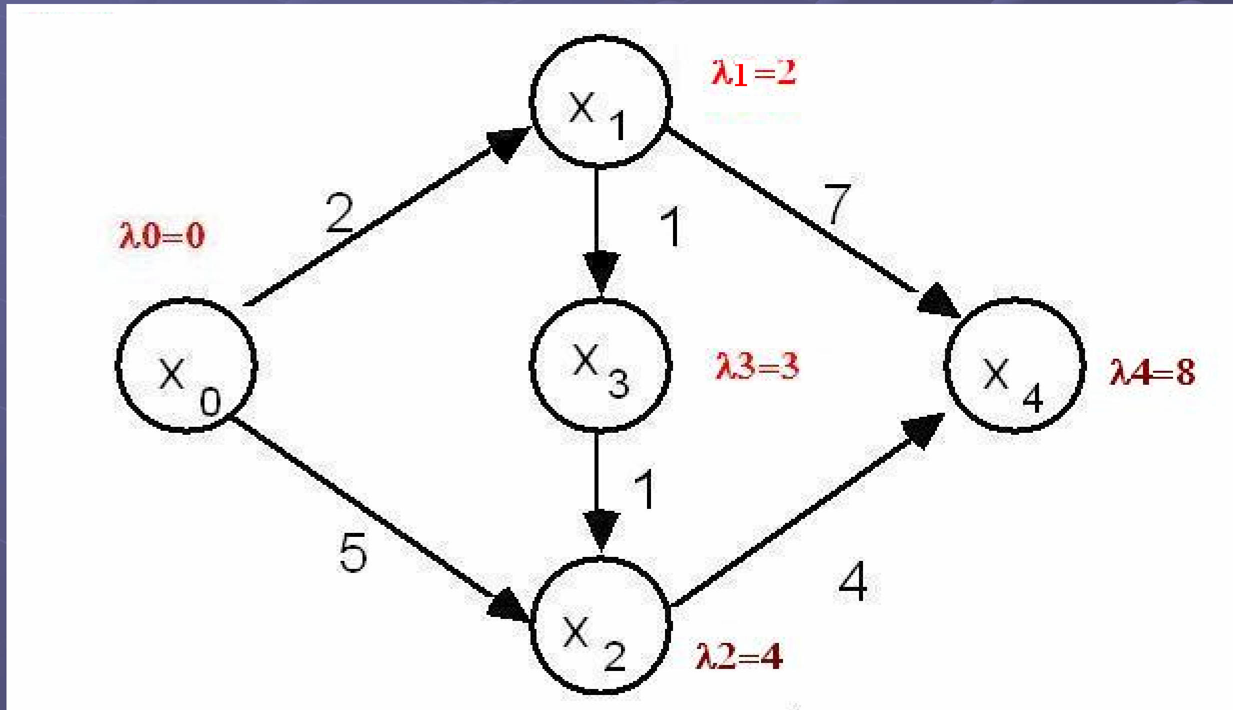
DIJKSTRA Algorithm
an example

End of the forth iteration



DIJKSTRA Algorithm
an example

End of the last iteration



Validity and complexity of Dijkstra algorithm

Lemma 4

Dijkstra algorithm is of complexity $O(n^2)$.

Theorem 3

λ_i obtained at the end of the algorithm are the shortest path values from x_0.

Bellman algorithm

Algorithm:

(i) **enumerate** all vertex of the graph, set $\lambda_0 = 0$.

(ii) for j = 1 to n – 1 set : $\lambda_j = \min(\lambda_k + v_{kj})$ over the set of predecessors x_k of x_j.

Theorem 4:

Bellman algorithm computes the shortest path values λ_i from x_0 in O(m).

Some path problems

- The longest path computation problem;
- The maximum probability path;
- The maximum capacity path value;

  - Exercise : compute the shortest path among these of maximum capacity.

Exercise: The itinerary of Michel Strogoff (from ROSEAUX)

Leaving from Moscow, Michel STROGOFF, courier of the tsar, was supposed to reach IRKUTSK. Before leaving, he had consulted a fortune teller who told him, amongst other things : "After KAZAN beware of the sky, in OMSK beware of the tartars, in TOMSK beware of the eyes, after TOMSK beware of water and, above all, always be careful of a large brown-haired person with black boots. " STROGOFF had therefore written on a map his "chances" of success for each route between two towns : these chances were represented by a number between 1 and 10 (measuring the number of chances of success out of 10). Ignoring probability calculation, he had therefore chosen his route by maximising the total sum of the chances.

The numbers of the cities are: MOSCOW (1), KAZAN (2), PENZA(3), PERM (4), OUFA (5), TOBOLSK (6), NOVO-SAIMSK (7), TARA (8), OMSK (9), TOMSK (10), SEMIPALATINSK(11), IRKOUTSK (12).

1. Determine the route of Michel Strogoff.
2. What was the probability, with the assumption of the independence of the random variables, that Strogoff would succeed?
3. What would have been his route if he had known the principles of probability calculation?

Shortest path algorithms and applications to networks

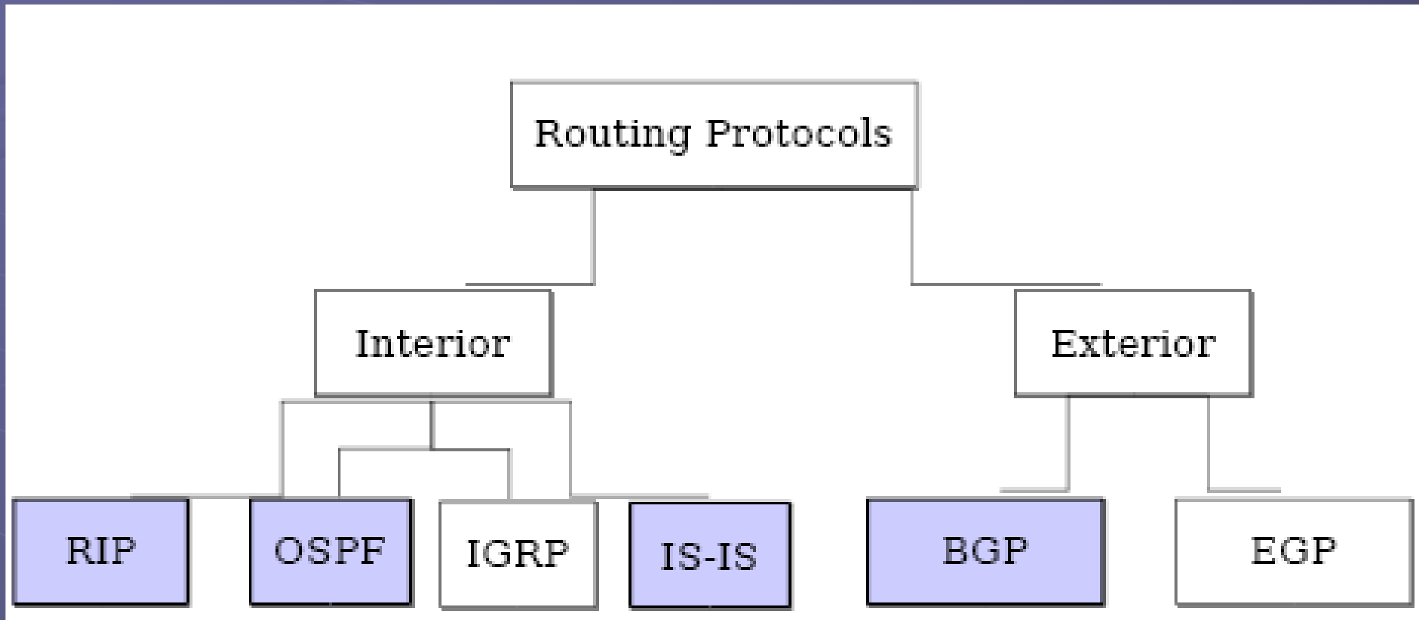
Routing protocols are implemented in a distributed way in IP networks.;

What is routing …

What is routing?

- The term **routing** corresponds to the mechanisms used by a host to transfer data to its destination by examining the information in the data.
- **Routing** is a key element of level **network** of TCP/IP stack. It uses information stocked in routing tables in each node-router.
  - The routing table stores the routes (and in some cases, metrics associated with those routes) to particular network destinations. It is frequent that in a routing table we find only the information about the gateway number toward the destination and not the entirely route.

Routing Protocols in Internet



Two main groups:

– Distance-Vector protocols: RIP, IGRP, BGP.

– Link-State protocols: OSPF, IS-IS

Routing Information Protocol (RIP) (RFC 2453)

Let D(i,j) represent the metric of the best route from entity i to entity j. It should be defined for every pair of entities. d(i,j) represents the costs of the individual steps. Formally, let d(i,j) represent the cost of going directly from entity i to entity j. It is infinite if i and j are not immediate neighbors. Since costs are additive, it is easy to show that the best metric must be described by:

$$D(i,i) = 0, \text{ all } i$$

$$D(i,j) = \min_k [d(i,k) + D(k,j)], \text{ otherwise}$$

and that the best routes start by going from i to those neighbors k for which d(i,k) + D(k,j) has the minimum value.

Implementing RIP

The **Routing Information Protocol** is a dynamic routing protocol used in local and wide area networks. As such it is classified as an interior gateway protocol (IGP) using the distance-vector routing algorithm.

Each router keeps a distance table for all destinations in the network. This table stores all shortest distance to any destination and the next neighbor to reach each of them according to the distance.

Periodically, each router announces its distance table to its direct neighbors;

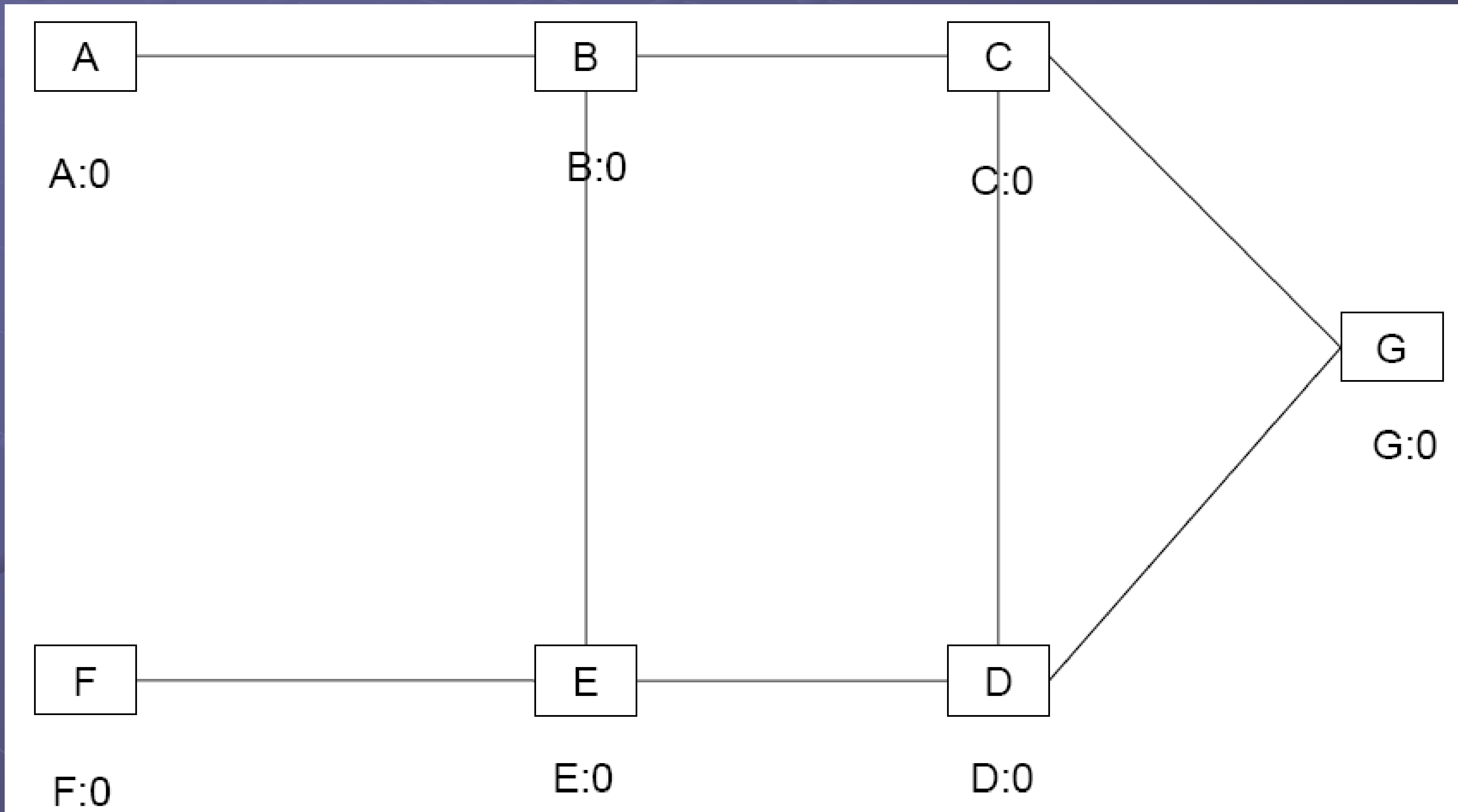
Any time some update is announced from a neighbor, do:

compute the new distance D';

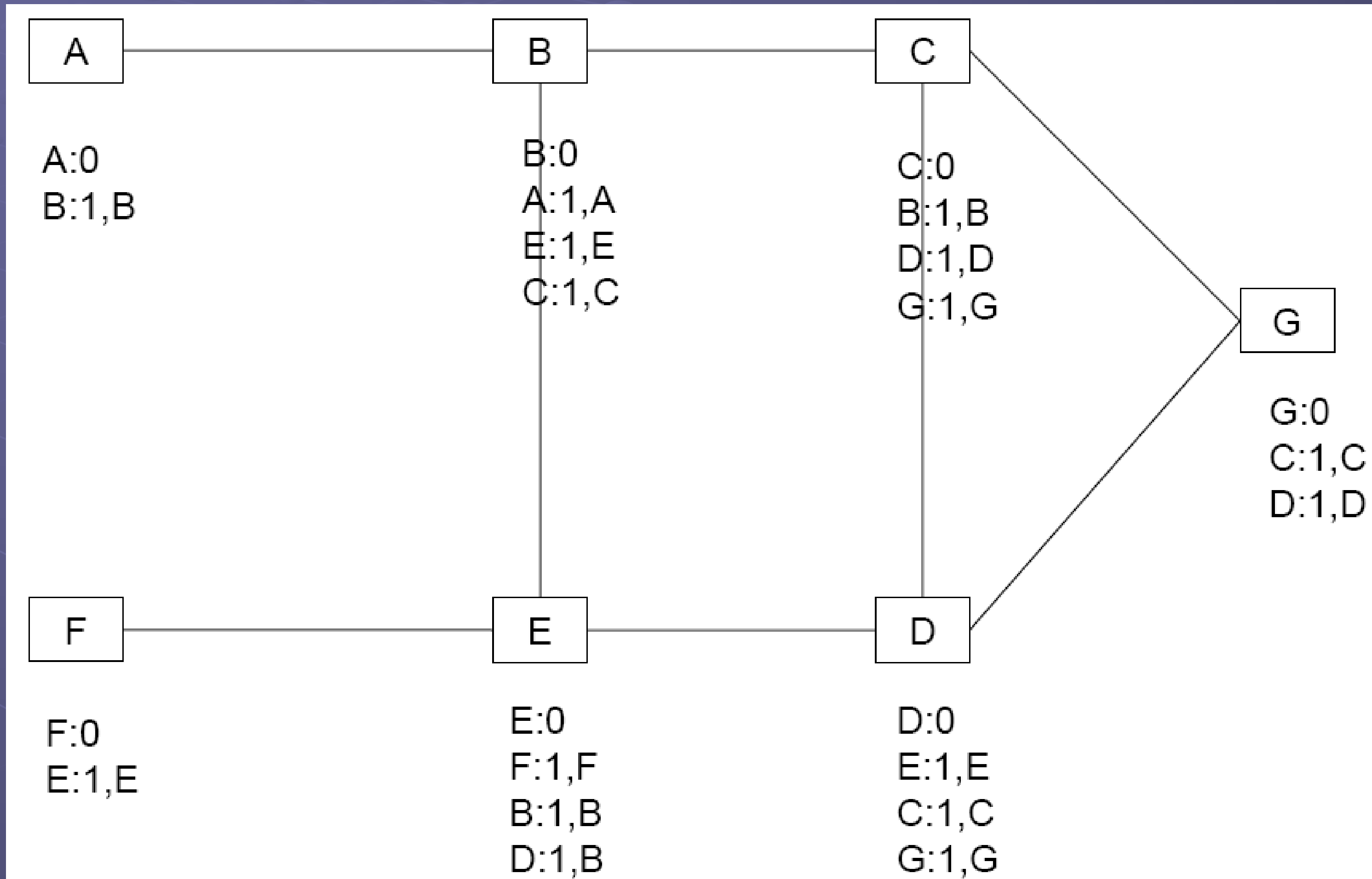
if D' < D keep the new value and the neighbor announcing it;

The update procedure is in origine of some limitations of the protocol…

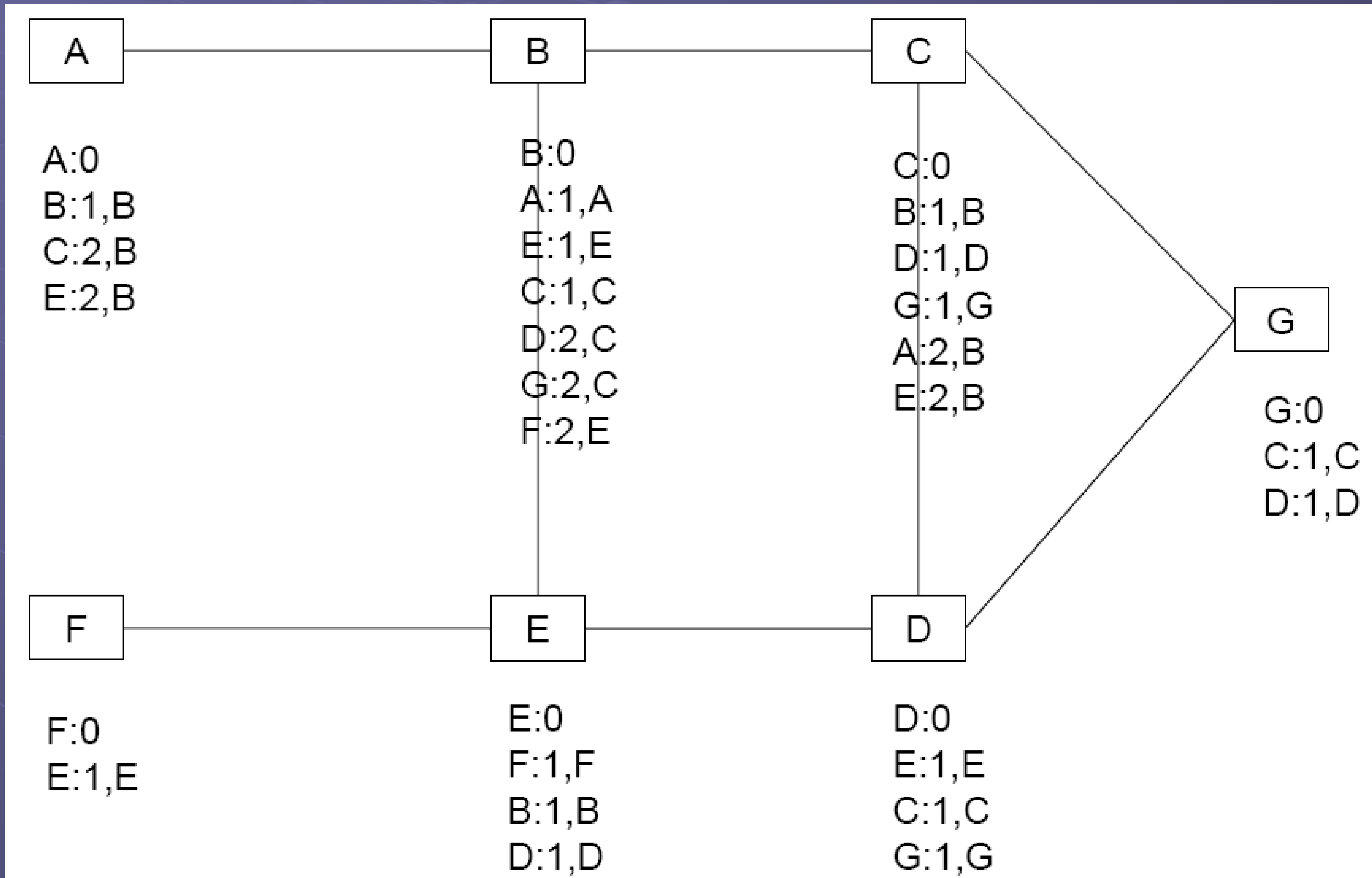
RIP: how it works?



RIP


A
B
C
G
F
E
D
A:0
B:1,B
B:0
A:1,A
E:1,E
C:1,C
C:0
B:1,B
D:1,D
G:1,G
G:0
C:1,C
D:1,D
F:0
E:1,E
E:0
F:1,F
B:1,B
D:1,B
D:0
E:1,E
C:1,C
G:1,G

RIP


A
B
C
G
F
E
D
A:0
B:1,B
C:2,B
E:2,B
B:0
A:1,A
E:1,E
C:1,C
D:2,C
G:2,C
F:2,E
C:0
B:1,B
D:1,D
G:1,G
A:2,B
E:2,B
G:0
C:1,C
D:1,D
F:0
E:1,E
E:0
F:1,F
B:1,B
D:1,D
D:0
E:1,E
C:1,C
G:1,G

Link-State protocols
OSPF (Open Shortest Path First)

- Principle:
  - **All nodes do have a map of the entire network**.
    - Determining the neighbors of each node
    - Distributing the information for the map (flooding)
    - Creating the map
  - **Computing the shortest paths**
    - Each node independently runs an algorithm (generally Dijkstra's algorithm is used) over the map to determine the shortest path from itself to every other node in the network.

Introduction to linear programming

- **Linear Programming**
  - **linear programming** (LP) is a technique for optimization of a linear objective function of variables x_1, x_2, …x_n, subject to linear equality and linear inequality constraints.

- **How to solve linear programming :**
  - The simplex algorithm (1951, 1963), developed by George Dantzig, solves LP problems by constructing an admissible solution at a vertex of the polyhedron and then walking along edges of the polyhedron to vertices with successively higher values of the objective function until the optimum is reached. (CPLEX, EXPRESS-MP, etc.).
  - **Alternative methods :**
    - the ellipsoid method by Leonid Khachiyan in 1979
    - In 1984, N. Karmarkar proposed a new interior point projective method for linear programming. (Karmarkar's algorithm)

Introduction to integer linear programming

- **Integer Linear Programming (ILP)**
  - An integer linear program is a linear programming problem with variables taking values in Z.
  - **Binary or 0-1 linear programming problems are a special case.**

- **How to solve integer linear programming :**
  - **Branch and bound methods, branch and cut…**

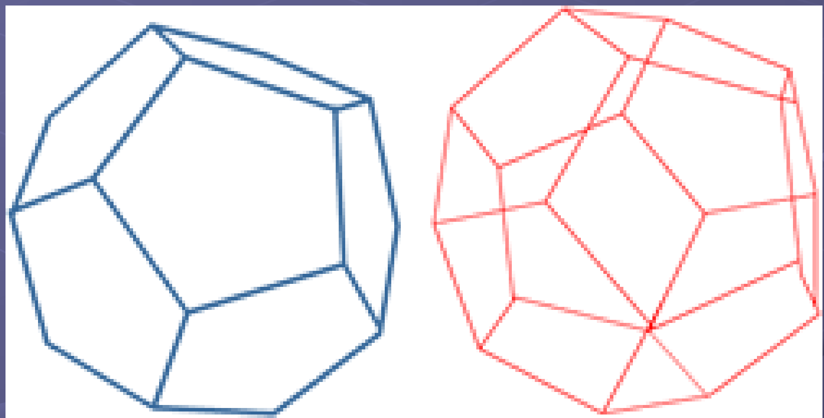
Dealing with the TSP problem

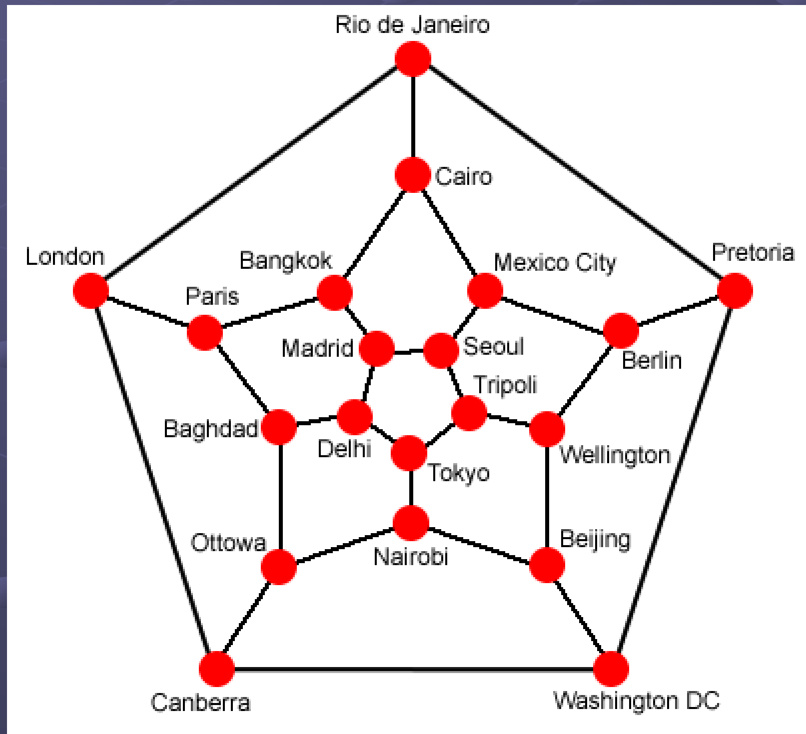
History

- **19th century**
  - The first methods are proposed by Sir William Rowan Hamilton et Thomas Penyngton Kirkman.
- **1930**
  - The TSP has been deeply studied by Karl Menger à Harvard.
- **1954**
  - Solution for TSP with 49 cities by Dantzig, Fulkerson et Johnson.
- **1975**
  - Solution for TSP with 100 cities by Camerini, Fratta et Maffioli
- **1987**
  - Solution for TSP with 532 cities and next with 2392 cities par Padberg and Rinaldi
- **1998**
  - Solution for TSP with 13 509 cities of US.
- **2001**
  - Solution for TSP with 15112 cities of Germanyt by Applegate, Bixby, Chvàtal and Cook from universities of Rice and Princeton.

The TSP problem

In 1859, the mathematician Sir W. R. Hamilton built a puzzle dodecahedron in wood. This dodecahedron has 20 vertices and 12 faces:





Find a Hamiltonian circuit.

The TSP problem, mathematical formulation

- ILP Formulation:
  - To each arc (i,j), associate variables x_{ij} taking 1 if included in the circuit and 0 otherwise.

$$Min \quad \sum_{i,j} d_{i,j} x_{i,j}$$

$$\sum_{j} x_{i,j} = 1 \qquad i = 1, \dots, n \quad (a)$$

$$\sum_{i} x_{i,j} = 1 \qquad j = 1, \dots, n \quad (b)$$

$$x_{i,j} \in \{0,1\} \qquad \forall i, \forall j.$$

$$\sum_{i \in S, j \in \bar{S}} x_{i,j} \geq 1 \quad \longrightarrow \quad u_i - u_j + n x_{i,j} \leq n-1 \quad 2 \leq i \neq j \leq n, \quad u_i \in N; \quad (c)$$

A naive method

- Let $P = Ax \le b$, ($x \ge 0$, et $A \ge 0$) a max. problem (cx, $c \ge 0$), and x^0 a continue solution of relaxed problem.
  - $\lfloor x^0 \rfloor$ is a feasible solution, $\lceil x^0 \rceil$ no.

- In general « rounding » the relaxed solution can lead to feasible or unfeasible solutions.
  - Example:

Maximize $x_1 + 2x_2$

$4x_1 + 3x_2 \le 12;$

$-2x_1 + x_2 \le 1;$

$3x_1 \le 5;$

$x_1 \ge 0$ *et* $x_2 \ge 0$, x_1, x_2 *integers.*

Relaxed solution (0.9, 2.8)

Integral solution (1, 2);

Exact resolution methods

- *Branch and Bound method (B&B).*
  - *Principle : build a research tree where the initial node corresponds to the relaxed problem (min).*
  - *Decompose the problem to sub-problems : the optimal solution should be in one of these sub-problems.*
    - *Any infeasible problem should be removed;*
    - *If possible, find the exact solution.*
    - *Otherwise find a lower bound , if it is larger than the best obtained solution the sub-problem should be discarded.*
    - *For the remaining cases, re-decompose the problem...*

Exact resolution methods

- .
- *Dakin Procedure (1965), and Lang & Doig (1960)*

  $$\text{Max } cx = z,$$
  $$Ax = b$$
  $$x_j \in N,\ j=1..n$$

- *Initiation : initial arborescence node S0*
  - *Solve the associated relaxed problem. If the obtained solution is integer, END.*
  - *Otherwise, an evaluation by excess is obtained; separate the problem again.*

Separation and Evaluation

- *Separation :*
  - *Separate on continuous variable x_k : $x_k \leq \lfloor x_k \rfloor$ and $x_k \geq \lfloor x_k + 1 \rfloor$ which gives two nodes S' et S'';*
- *Evaluation by excess :*
  - *Solve the two « continuous » linear programs:*
    - $PL_{S'} = PL_S + x_k \leq \lfloor x_k \rfloor$;
    - $PL_{S''} = PL_S + x_k \geq \lfloor x_k + 1 \rfloor$;

Separation and evaluation

- *Node choice*
  - *Depth first*

- *Default evaluation*
  - *Best known solution z_0.*

- *Abandon the search on S if $v(S) \leq v(z_0)$.*
  - *Stop when all nodes are abandoned.*

Linear programming modeling

General case

Suppose that a linear programming is composed of :

- The objective function of ***n*** variables x_j (*maximization)* ;
- All variables take positive values.
- The constraints are linear functions bounded by some constant

That is :

Interpretation :

n activities, m resources…

$$Max \quad z = \sum_{j=1}^{n} c_j x_j$$

$$\begin{aligned}
&s.c.\ \ A_{11}x_1 + A_{12}x_2 + \ldots + A_{1n}x_n \le b_1;\\
&A_{21}x_1 + A_{22}x_2 + \ldots + A_{2n}x_n \le b_2;\\
&\ldots\\
&A_{m1}x_1 + A_{m2}x_2 + \ldots + A_{mn}x_n \le b_m;\\
&x_{1,}x_{2,}\ldots,x_{n,} \ge 0.
\end{aligned}$$

Simplexe algorithm: geometrical interpretation

maximize: $Z = 2x_1 + x_2$

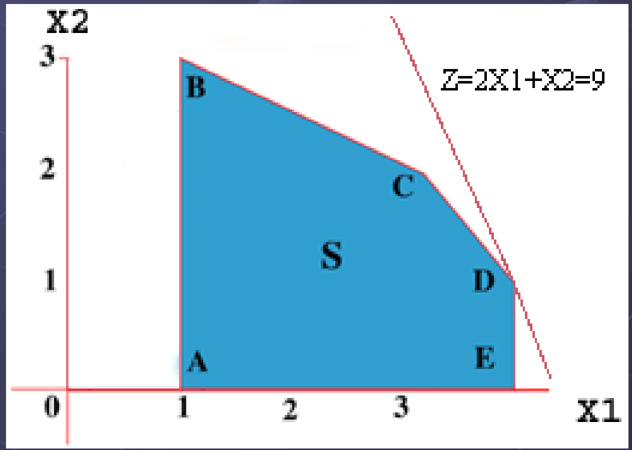
Subject to: $x_1+2x_2 \leq 7$

$x_1+x_2 \leq 5$

$x_2 \geq 1$

$x_1 \leq 4$

$x_i \geq 0$



Linear programming modeling

Modeling : importance of linearity;

- Linearity of objective function:
  - maxmin or minmax functions
  - With absolute values
  - Objective function « rapport » or product
- Linearity of constraints :
  - Capacity constraints
  - Demand constraints
  - Mass balance constraints
  - Proportion constraints

LP modelling

Example. Look at the case of production problem where one should find the quantities to be produced under capacity constraints. Suppose that some factory produces two products A and B, each of them passing through cutting process (C) and refinement (R) :

process	**Cutting**	Refinement
Time for processing A	2 hours	3 hours
Time for processing B	2 hours	1 hour
Maximal capacity	200 hours	100 hours

Knowing that the profit for each product A and B are 20 € and 10 €, find the quantities to be produced in order to maximize the profit ?

LP Modelling

Three TV models A, B et C providing profits of 160, 300 et 400 francs. Each TV requires some time Fi for processing pieces, some time Ai for assembling and some Ei time for refining. In one week there is 150 available hours for processing, 200 for assembling and 60 refining. Propose an LP model for the production planning that maximizes the overall profit.

	A	**B**	**C**	
Fi	3	3,5	5	**150**
Ai	4	5	8	**200**
Ei	1	2,3	3	**60**

Linear programming modeling

Modeling examples :

lower and upper bounds;
render inequalities to equalities;
express unsigned variables as nonnegative ones;
express the absolute value;
express minmax and maxmin functions…

Logic constraints:

If A then B; A iff B; If A not B; If not A then B; If B and C then A;
if two or more variables in {B, C, D, E} then A;
If M or more over N variables B, C, D, …, then A;

Modeling by binary variables

At most one of A, B,...,H	$a+b+c+d+e+f+g+h \leq 1$
Exactly two of A, B,...,H	$a+b+c+d+e+f+g+h = 2$
If A then B	$b \geq a$
Not B	$\bar{b} = 1 - b$
If A then not B	$a + b \leq 1$
If not A then B	$a + b \geq 1$
If A then B, and if B then A	$a = b$
If A then B and C	$b \geq a$ and $c \geq a$
If A then B or C	$b + c \geq a$
If B or C then A	$a \geq b$ and $a \geq c$
	or alternatively: $a \geq \frac{1}{2} \cdot (b + c)$
If B and C then A	$a \geq b + c - 1$
If two or more of B, C, D or E then A	$a \geq \frac{1}{3} \cdot (b + c + d + e - 1)$
If M or more of N projects (B, C, D, ...) then A	$a \geq \frac{b+c+d+\ldots-M+1}{N-M+1}$

Linear programming modeling

Modeling examples :

disjunctive constraints :

$\text{Constraint}_1 \geq b_1$ or $\text{Constraint}_2 \geq b_2$, b_1, $b_2 > 0$;

conjunctive constraints;

task A before task B, or task A before B and C...

Express in a linear form the following model:

$$\text{Min } \{(c^T x + d)/(f^T x + g) \mid Ax \leq b,\ f^T x + g > 0,\ x \geq 0\};$$

LP modelling : Exercices

- …

Introduction to computational complexity theory

- Complexity theory deals with two aspects:
  - Algorithm's complexity.
  - Problem's complexity.

- References
  - S. Cook, « The complexity of Theorem Proving Procedures », 1971.
  - Garey et Johnson, « Computers and Intractability, A guide to the theory of NP-completeness », 1979.
  - J. Carlier et Ph. Chrétienne « Problèmes d'ordonnancements : algorithmes et complexité », 1988.

Basic Notions

- Some problem is a “question” characterized by parameters and needs an answer.
  - Parameters description;
  - Properties that a solutions must satisfy;
  - An instance is obtained when the parameters are fixed to some values.
- An algorithm: a set of instructions describing how some task can be achieved or a problem can be solved.
- A program : the computational implementation of an algorithm.

Algorithm's complexity (I)

- There may exists several algorithms for the same problem
- Raised questions:
  - Which one to choose ?
  - How they are compared ?
  - How measuring the efficiency ?
  - What are the most appropriate measures, running time, memory space ?

Algorithm's complexity (II)

- Running time depends on:
  - The data of the problem,
  - Quality of program...,
  - Computer type,
  - Algorithm's efficiency,
  - etc.
- Proceed by analyzing the algorithm:
  - Search for some n characterizing the data.
  - Compute the running time in terms of *n*.
  - Evaluating the number of elementary operations, *(elementary operation = simple instruction of a programming language).*

Algorithm's evaluation (I)

- Any algorithm is composed of two main stages: initialization and computing one
- The complexity parameter is the size data n (binary coding).

Definition:

Let be $n>0$ and $T(n)$ the running time of an algorithm expressed in terms of the size data n, $T(n)$ is of $O(f(n))$ iff $\exists\, n_0$ and some constant c such that:

$\forall n \geq n_0$, we have $T(n) \leq c\, f(n)$.

If f(n) is a polynomial then the algorithm is of polynomial complexity.

- Study the complexity in the worst case.
- Study the complexity in average : $tm(n)$ is the mean value of execution time when the data and the associated distribution law are given.

Algorithm's evaluation (II) example

Given N numbers a_1, a_2, .., a_n in {1,…, K}.

The algorithm MIN finds the minimum value.

<u>Algorithm MIN</u>

```
Begin
  for i=1 to n read a_i;
  B:=a_1, j:=1;
  for i=2 to n do :
          if B=1 then j:=n;
          elseif a_i < B then
                  begin
                          j:=i;
                          B:=a_i;
                  end;
  write : the min value is a_j;
End.
```

Algorithm's evaluation (III)

- Study the complexity of algorithm MIN.

Proposition 1. The complexity of Algorithm MIN is in O(n).

Proposition 2. if numbers a_i are independent random values and if any a_i has some probability 1/K to be equal to 1, the complexity in average of algorithm MIN, apart the reading data phase, is in O(1).

Importance of polynomial algorithms (I)

f(n)	n=10	n=20	n=30	n=40	n=50
n	0.00001 sec	0.00002 sec	0.00003 sec	0.00004 sec	0.00005 sec
n^2	0.0001 sec	0.0004 sec	0.0009 sec	0.0016 sec	0.0025 sec
n^3	0.001 sec	0.008 sec	0.027 sec	0.064 sec	0.125 sec
2^n	0.001 sec	1 sec	17.9 min	12.7 days	35.7 years
3^n	0.059 sec	58 min	6.5 years	3.855 centuries	$2*10^8$ centuries

An elementary operation is run in one microsecond.

Importance of polynomial algorithms (II)

f(n)	Todays computers	100 times faster	1000 times faster
n	N1	100 N1	1000 N1
n^2	N2	10 N2	31.6 N2
n^3	N3	4.64 N3	10N3
2^n	N4	N4 +6.64	N4 + 9.97
3^n	N5	N5 + 4.19	N5 + 6.29

Problem's sizes solved in one hour run time

Computational complexity theory

- The decision problem is some mathematical question requiring some answer yes or no.
- Computational Complexity Theory is concerned with the question: for which decision problems do efficient algorithms exist

SAT

- Satisfiability is the problem of determining if the variables of a given boolean formula can be assigned in such a way as to make the formula evaluate to TRUE.
- In complexity theory, the Boolean satisfiability problem (SAT) is a decision problem, whose instance is a Boolean expression written using only AND, OR, NOT, variables, and parentheses.
- The question is: given the expression, is there some assignment of *TRUE* and *FALSE* values to the variables that will make the entire expression true?
- A formula of propositional logic is said to be *satisfiable* if logical values can be assigned to its variables in a way that makes the formula true.

Travelling salesman problem

- Given a weighted graph G=(X,E,v)

  X = Vertices (= Cities)
  E = Edges (pair of cities)
  v = Distances between cities

- *Find the shortest tour that visits all cities*

Partition problems

- Partition problem
  - Data: given a set $A=\{a_i \mid i\in I\}$ of n integer numbers.
  - Question : is-there some partition of A in two subsets A1 and A2 of equal weight ?

- Tripartition problem
  - Data: given a set $A=\{a_i \mid i\in I\}$ of $n=3q$ integer numbers such that $\sum_{i\in I} a_i=qB$ and $B/4<a_i<B/2$ and B some positive integer.
  - Question : Is-there some partition of A in q subsets of cardinal 3 and weight B?

Some equivalent problems

- Vertex covering.
  - Data : a graph $G=(V,E)$ with V a set of vertex, E a set of edges and some positive integer $B \leq |V|$.
  - Question : Is-there some subset $V' \subseteq V$ such that $|V'| \leq B$, and for each edge $(i,j) \in E$, $i \in V'$ or $j \in V'$?
- Independent set
  - Data : a graph $G=(V,E)$ with the set of vertex *V and E* the set of edges and some positive integer $B \leq |V|$.
  - Question : Is-there some $V' \subseteq V$ $|V'| \geq B$ such that for any $(i,j) \in E$, $i \notin V'$ or $j \notin V'$?
- Maximal clique.
  - Data : a graph G = (V,E) where V is the vertex set and E the set of edges, and a positive integer B.
  - Question : Is-there some $V' \subseteq V$ such that the corresponding sub-graph is complete and of size greater or equal to B?

Scheduling problems

- One machine problem
  - Data : a set I of n independent and indivisible tasks; for each task $i \in I$ we have its duration p_i, availability date r_i and its deadline d_i.
  - Question : is-there some scheduling ϕ of these n tasks one a single machine that satisfies the availability and deadline dates?
- Two processors problem
  - Data : a set I of n independent and indivisible tasks , durations p_i and a positive integer B.
  - Question : Is-there a scheduling ϕ of these n tasks on two processors of duration less or equal to B ?
- NTRM
  - Data : a set $I=\{1,2,.. n\}$ of tasks of durations 1 and deadlines d_1, d_2, .., d_n, a partial order $<$ on I, and a positive integer B.
  - Question : Is-there a scheduling ϕ of these tasks on one machine satisfying the partial order and such that the number of delayed tasks is $\leq$ B.

Complexity theory: basic notions

- Why using the decision problems?
  - To introduce a simple formalism and facilitating the comparison between problems.
- *The complexity theory relies on Turing Machine…*
  - ….

The class NP

- Alternatively:
- We distinguish the following complexity classes:
  - Class P : some problem is in P if it can be solved in polynomial time to the size of data by a determinist algorithm.
  - A problem is said to be in **NP** if and only if for a guessed solution there exists a polynomial time algorithm verifying the solution.
  - *Class NP : it groups all decision problems such that an answer yes can be decided by a non-determinist algorithm in polynomial time to the size of data.*
    - **Polynomial time verification**

NP-completeness

Paper of Stephen Cook, «The complexity of Theorem Proving Procedures», 1971.

- Defines the polynomial reduction
- Defines the decision problems and the class NP.
- Shows that the problem SAT is at least as difficult as all the others in NP → NP-complete

Complexity theory
polynomial reduction

- Polynomial reduction allows to compare NP problems in terms of computational complexity
- *Definition: P_1 is polynomially reduced to P_2 ($P_1 \propto P_2$) if P_1 is polynomial there exists a polynomial algorithm A that builds for any d_1 of P_1 some data d_2 of P_2 such that d_1 has answer YES iff d_2=A(d_1) has answer YES.*
- Some problem is said NP-Complete if he is in NP, and any NP-Complete problem can be polynomially reduced to this problem.
- The polynomial reduction defines a pre-order relation on NP.
- Cook'sTheorem : SAT is NP-Complete.

NP-completeness demonstration techniques (I)

- The general method:
  - 1) show first that $\Pi \in NP$
  - 2) show that there exists P'$\in$NP-complete such that P'$\propto \Pi$.

- The following problems are NP-complete.
  - TSP,
  - Partition problems,
  - Exact cover
  - Clique

Demonstration techniques (II)

- Three main techniques are used to show the NP-completeness of combinatorial problems.
  - Restriction
    - examples : minimal covering, knapsack, etc.
  - Local replacement
    - examples : 3SAT, X4C…
  - Component design
    - example : NTRM…

Demonstrating the NP-completeness

- Show the three following propositions:
  - Proposition 1. The two-processors problem is NP-complete.
    - *show **partition** $\propto$ two-processors.*
  - Proposition 2. The one machine problem is NP-complete.
    - ***partition** $\propto$ one machine problem.*

Exercises

1) Show that the knapsack problem is NP-complete.

- Data : A finite set X, for any $x_i \in X$, there is some weight $w(x_i)$ and profit $p(x_i)$, and B and K two positive integers.
- Question : Is–there some $X' \subseteq X$ such that $\sum_{xi \in X'} w(x_i) \leq B$ and $\sum_{xi \in X'} p(x_i) \geq K$?

2) Show the NP-completeness of the following problems:

- Data : a graph G=(V,E) and some positive number $K \leq |V|-1$.

- Question : is there some spanning tree of G with all vertex degrees less or equal to K; in other words: some subset $E' \subseteq E$ such that $|E'|=|V|-1$, and G'(V,E') is connected and no vertex is incident to more than K arcs ?

suggestion : TSP

NP-completeness : conjecture

- Fundamental Conjecture: P≠NP.

You win 1000000 USD if You show that P=NP or P≠NP.

Pseudo-polynomial algorithms (I)

- Dynamic programming
  - Idea : breaking down the initial problem in a sequence of simpler problems, solving the n-the problem can be done by recurrence on this of (n-1)-the one.
  - WESS problem (weight of a subset)
    - Data : a finite set A composed of n elements $a_i \in Z+$ and a positive integer K.
    - Question : is-there a subset of A of weight K ?

Pseudo-polynomial algorithms (II)

<u>Algorithm WESS</u>

```
Begin
  for k=0 to Σ_{ai∈A} a_i
      for j=1 to n do :
      WEIGHT(k,j):=false;
       end for;
  end for;
   set WEIGHT(0,0) := true;
  for j=1 to n
       for k=0 to Σ_{ai∈A} a_i do :
       if (k≥a_j and WEIGHT(k-a_j, j-1)=true) or WEIGHT(k, j-1)=true
              then WEIGHT(k, j)=true;
       end for;
  end for;
end.
```

<u>Proposition</u> : Algorithm WESS is of complexity $O(n\sum_{a_i \in A} a_j)$ and assigns true to WEIGHT(K, n) if there exists some subsets of n numbers $a_1, a_2, .., a_n$ of weight K.

Exercise: AN INVESTMENT PROBLEM

1) An example. 6 million is at our disposal, to invest in 3 regions. The following table shows the benefits given by the invested sums.

	Region I	Region II	Region III
1 million	0.2	0.1	0.4
2 million	0.5	0.2	0.5
3 million	0.9	0.8	0.6
4 million	1	1	0.7

Determine the optimal investment policy for the three regions using a "dynamic programming" method. The idea is to associate a graph with levels to the data. Level 0 contains only the vertex (0,0), (because no money has been invested yet). Level 1 contains the vertices (1,0) (1,1) (1,2) (1,3) (1,4), which correspond to the cumulated amounts invested in region 1. Level i contains the vertices (i, 0), (i, 1), (i, 2), (i, 3), (i, 4), (i, 5), (i, 6), which correspond to the sums invested in the regions 1 .. i (i = 2, 3). The arcs are placed between the levels i and i +1, valuated by the sums invested in the region i +1. The last vertex is (3,6). The goal is to seek a maximum value path in this graph.

2) The general case. More generally, we have B million to invest in n regions. We shall set fi(y), the optimal profit for a cumulative investment of a sum y in the regions 1, 2, .., i. We have f0(0) = 0. Determine a recurrence formula connecting fi to fi-1 for i from 1 to n.

3) What is the complexity of the dynamic programming method in function of n and B, and complexity of enumerating all possible solutions.

Strong NP-completeness (I)

- *Binary code* is the system of representing text or computer processor instructions by the use of the *Binary* number system's two-*binary* digits "0" and "1".
- Binary code of a → $\log_2(a)$ places
  - example : the size of 2^{31} is $\log_2(2^{31}) + 1 = 32$, thus we need 32 bits to code it in machine;
- In computational complexity theory, a numeric algorithm runs in pseudo-polynomial time if its running time is polynomial in the *numeric value* (unary code) of the input.
- An NP-complete problem with known pseudo-polynomial time algorithms is called weakly NP-complete. An NP-complete problem is called strongly NP-complete if it is proven that it cannot be solved by a pseudo-polynomial time algorithm.

Strong NP-completeness (II)

- Definition : some problem is strongly NP-complete, if it is NP-complete and if the existence of a solution pseudo-polynomial algorithm yields this of a polynomial algorithm.
  - Examples : TSP, tripartition, clique, sat.
  - What about partition, two-processors ?
  - Proposition. The one machine problem is strongly NP-complete.