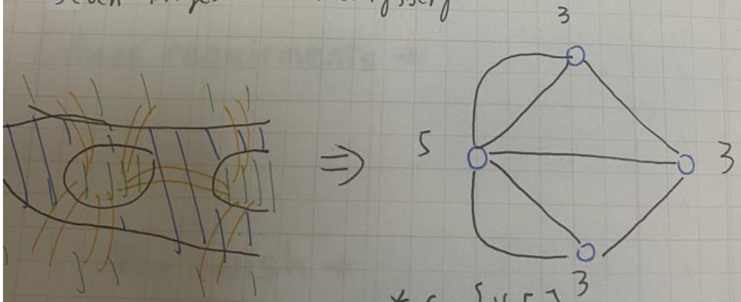
Seven Brige of konigsberg 七橋問題

* konigsberg in Prussia
  - kaliningrad, Russia
  - Pregel River
  - two large islands
  - Seven briges

* Leohard Euler [1735] 尤拉
  - Find a walk through the city that would cross each brige once and exactly one
  - The first paper in the history of graph theory

Seven Briges of konigsberg



* $G = \{V, E\}$
  - $V(G)$: Vertex set 地
  - $E(G)$: edge set 橋

Problems & Difficulties needing exploration

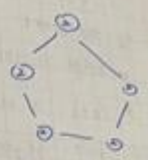
- degree

Number of edges

$$\sum_{v_i \in V(G)} degree(v_i) = |E(G)| \times 2$$

* Vertex types
  - odd or even degrees

* Eulerian path (trial?) / Euler walk
  - visit every edge exactly once
  - 0 or 2 nodes with odd degrees



~~What if 1 node with odd degree?~~

所有奇數不可能發生

* Eulerian circuit (cycle) / Euler tour
  - begin and end at the the same vertex
  - 0 node with odd dergess

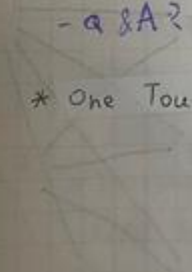
$\{$Euler walks$\} \supseteq \{$Euler tours$\}$

My Opinions

Thoughts, inspirations, and suggestions

* Seven brige problem
  - Q&A? NO

* One Touch Drawing

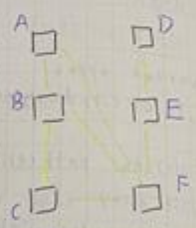


在對的時間，做對的事，
以表明對人的重視。
《紐基文教》

My Notes

Important Concepts worth keeping

Basic Terminologies

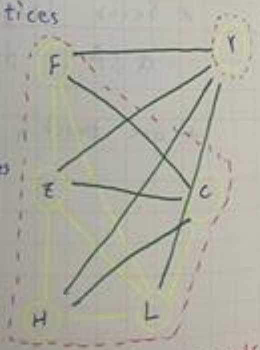


* Undirected graph
* Directed graph (digraph)
* Adjacent vertias
* Edge is incident to vertices
* Path: a sequence of edges
* cycle: begin & end at the same vertex

How many times to find old friend

6 degrees of separation from small world experiment

* simple path: a path that pases through any vertex only once
* Simple cycle: a cycle that pases through the other vertices only once
* connected Graph
  - There is a path between two vertices
  - Disconnected Graph
* Complete Graph $|E| = ?$
  - There is an edge between any two vertices



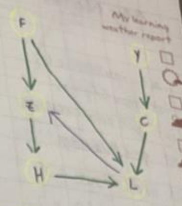
Connected components

"Do not worry about tomorrow; for tomorrow will care for itself. Each day has enough trouble of its own." (New Testament)

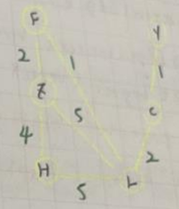
Two subgraphs

rr

* Strong connected graph (有方向
  - For any two vertices on a digraph, there is a path from one vertex to the other
  - Q&A | How to decide?



* weighted graph (有權重)
  - the edges have numeric labels



Graphs as ADTs

My Opinions
Thoughts, inspirations, and suggestions

* Variations of an ADTs are possible
  - Vertices may or may not contain values
    - Many problems have no need for vertex value
    - Relationships among vertices is what is importa
* Insertion and Deletion operations for graphs apply to vertices and edges.

* Graph can have traversal operations.

不要為明天憂慮
因為明天自有明天的憂
一天的難處一天當就夠
《新約

My Notes
Important Concepts worth keeping

Today:

```
int numVertices;
int numEdges;
int getNumVertices;
int getNumEdges;
void add(Edge e);
void remove(Edge e);
bool isEdge(vertex u, vertex v);
int getDegree(vertex v);
bool isConnected(Graph g);
EdgeList traverse(Graph g);
```

Graph Representations

* Most common implementations of a graph
  1. Adjacency matrix
  2. Adjacency list

* Adjacency matrix for a graph that has n vertices numbered 0, 1, ..., n-1
  - An n by array matrix such that matrix[i][j] indicates whether an edge exist from vertex i to vertex j

So in everything, do to others

My Questions

Problems & Difficulties needing exploration

Adjacency Matrix: examples

In-dergee vs. out degree

	P	Q	R	S	T	W	X	Y	Z
P	0	0	1	0	0	1	0	0	0
Q	0	0	0	0	0	0	1	0	0
R	0	0	0	0	0	0	1	0	0
S	0	0	0	0	1	0	0	0	0
T	0	0	0	0	0	1	0	0	0
⋮									
W	0	0	0	1	0	0	0	1	0
X	0	0	0	0	0	0	0	0	0
Y	0	0	1	0	0	0	0	0	1
Z	0	0	0	0	0	0	0	0	0

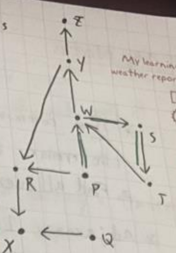
No predecessor

Traverse a path

Q&A: isEdge(u,v)!

Q&A: getDegree(v)?

traverse(g) O(|V|²)



My learning weather report

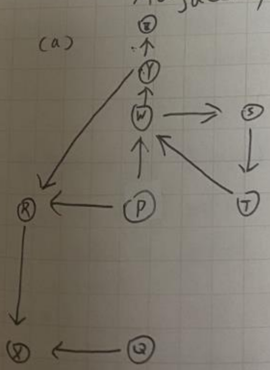
- For an unweigthed graph, matrix[i][j] is
  - 1 (or true) if an edge exists from vertex i to vertex j
  - 0 (or false) if no edge exists from vertex i to vertex j
- For a weigthed graph, matrix[i][j] is
  - The weight of the edge from vertex i to vertex j
  - ∞ (or 0) if no edge exists from vertex i to vertex j

My Opinions

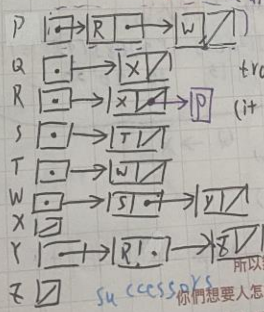
Thoughts, inspirations, and suggestions

Adjacency List: Example

(a)



(b) out-degree



successors

Q&A: isEdge

Q&A: getDegree

Q&A: undirected graph

traverse(g): O(|V|+|E|)

(if it is a sparse matrix

密碼
cipher key

所以無論何事，
你們想要人怎樣待你們，
你們也要怎樣待人。
《新約聖經》

My Notes
Important Concepts worth keeping

Graph Representations

Today: , ,

* Two common operation on graph isEdge(i,j)

1. Determine whether there is an edge from vertex i to vertex j
2. Find all vertices adjacent to a given vertex i

* Adjacency matrix getDegree(i,j)

- support operations 1 more efficiently

* Adjacency list

- support operation 2 more efficiently
- often requires less spaces than an adjacency matrix

Other Graph Representations

* Mapping from vertex label to array indices

P Q R S T W X Y Z
0 1 2 3 4 5 6 7 8

* sequential representation

- nodes + edges

[0]	[1]	[2]	[3]	…	[7]	[8]	[9]	[10]	[11]	[12]	[13]
10	12	13	14		19	20	20	2	5	6	6

Q → X

isEdge(u,v)

getEdge(v,)

undirected graph : |V| + 2|E| + 1

My Questions
Problems & Difficulties needing exploration

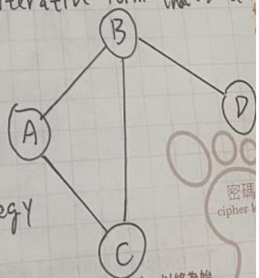
Graph Traversals

* Visited all the vertices that it can reach
* visits all vertices of the graph it and only if the graph is connected
  - A connected
    * the subset of vertices visited durning a traversal that begins at a given idex vertex
* To prevent indefinite loops (break the cycles)
  * Mark each vertex durning a visit, and
  * Never visit a vertex more than once

DFs and BFs Traversals

* Depth-First Search (DFS) Traversal
  - Proceeds a path from a vertex as deeply into the graph as possible before backing up
  - A "last visited, first explored" strategy (stack)
  - Has simple recursive
  Has an iterative form that use a s

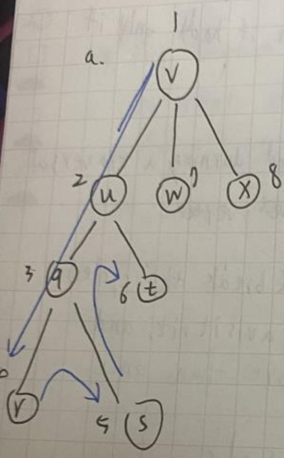
* Bread th-First (BFS) Traversal
  - visit every vertex adjacent to vertex v before visiting any other vertex
  - A "first visited, first explored" strategy
  - An iterative from use a queue
  - A recursive form it possible, but not simple



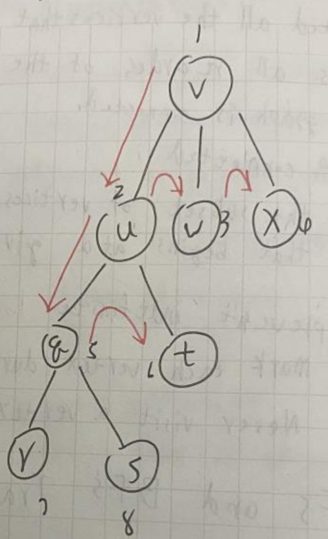
My Opinions
Thoughts, inspirations, and suggestions

My Notes

Important Concepts worth keeping DFS and BFS Traversals Today: , ,



DFS



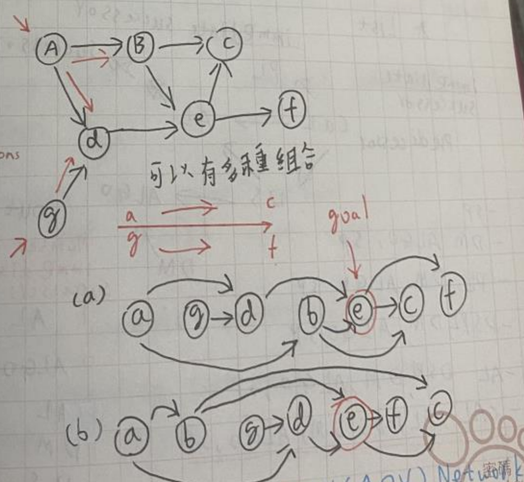
BFS

My Questions Topological Sort: Definition
Problems & Difficulties needing exploration

My learning weather report

* Topological order
  - A list of vertex in a directed graph without cycles (Acyclic Digraph or Directed Acyclic Graph DAG) such that vertex x preceds vertex y if there is a directed edge from x to y in the graph
  - several tpological orders are possible for a given graph
* Topological sorting
  - Arranging the vertices into a topological order

My Opinions
Thoughts, inspirations, and suggestions



Activity-on-vertex (AOV) Network

我們了解人人各承不同之稟賦，
其性格、能力與環境各異，
故充分發揮個人潛力就是成功。
《中原大學教育理念》

cipher key 密碼

My Notes
Important Concepts worth keeping

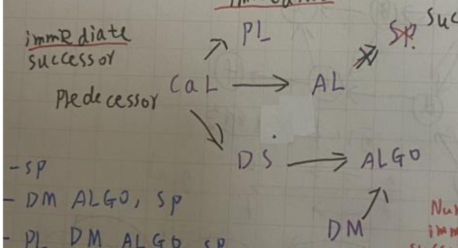
Topological sort : Algorithms

* top Sort 1

1. Find a vertex that has no successor (out-degree =0)
2. Add the vertex to the beginning of a list →fc
3. Remove that vertex from the graph, as well as all edges that lead to it
4. Repeat that previous steps until the graph is empty.
   - when the loop ends, the list of vertices will be in topological order.

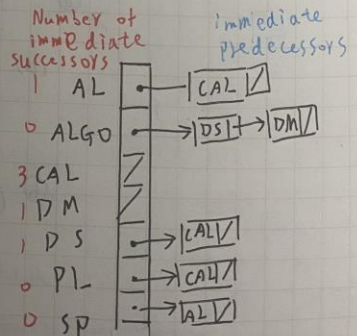
A trace of top sort 1 (concept)

* List



-SP
-DM ALGO, SP
-PL DM ALGO SP
-DS,PL,DM, ALGO,SP
AL DS,PL,DM,ALGO,SP
CAL, AL,PL,DS,DM,ALGO,SP

out - degree



If someone forces you to go one mile

My Questions Topological Sort : Algorithms
Problems & Difficulties needing exploration

My learning weather report

* top Sort 2
  - A moditication of the iterative DFS algothrim
  - push all vertices that have no predecessor on to a stack
  - Each time you pop a vertex from the stack, add it to the beginning of a list of vertices
  - When the traversal ends, the list of vertices will be the topological order.

Spanning Tree : Definition

* A tree is an undirected graph connected graph without cycle
* A spanning tree of a connected undirected graph G is

My Opinions
Thoughts, inspirations, and suggestions

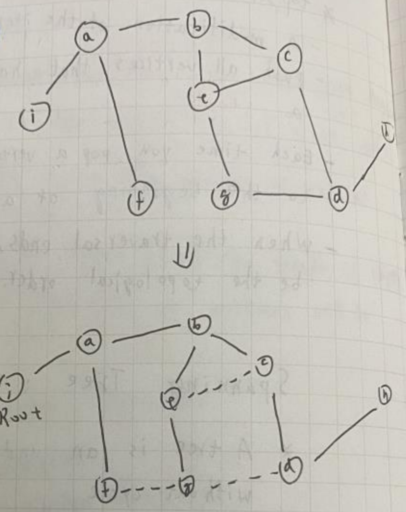
  - A subgraph of G that contains all of G's vertices and enough of its edge to form a tree
  - Application example : communication network
    - CISCO , spanning Tree protocol (STP)
    - connected ⟷ acylie

My Notes Spanning Tree : Definition

Important Concepts worth keeping

Today : / /

* To obtain a spanning tree from a connected undirected graph with cycles
  - Remove edges until there are no cycles



Spanning Tree : properties

* Detecting a cycle in a undirected connected graph
  - DFS or BFS
  - A connected undirected graph that has n vertex must have at least n-1 edges
  - A connected undirected graph that has n vertices and exactly n-1 edges cannot contain a cycle
  - A connected undirected graph has n vertices and more than n-1 edges must contain

TEAM:

Together Everyone A

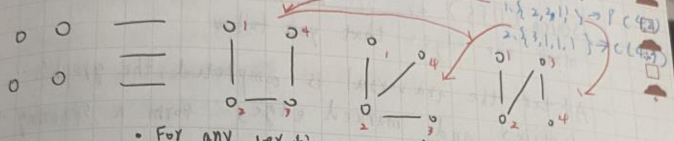
My Questions Partice : Number of spanning trees

Problems & Difficulties needing exploration

My learning weather report

* How many different spanning trees

- Two Graph G and H are isomorphic 同構 if and only if there is a bijection f between their vertex sets



- For any vertices x and y in G, there are adjacent in G iff f(x) and f(y) are adjacent in H

Counting Spanning Trees

various vertex labeling : n^{n-2}

* 2 nodes → $2^{2-2} = 1$

* 3 nodes → $3^{3-2} = 3$

* 4 nodes → $4^{4-2} = 16$

My Opinions

Thoughts, inspirations, and suggestions

* Prufer sequence

1. Each label tree with vertices has a unique Prufer sequence of length n-2

- conversion algorithms
  - Leaf with the smallest label
  - keep the label of its parent

2. Each Prufter sequence of length n-2 has ... with n vertices

密碼 cipher key

團隊成效可好可壞。

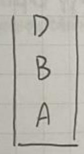
DFS / BFS for Spanning Trees ,

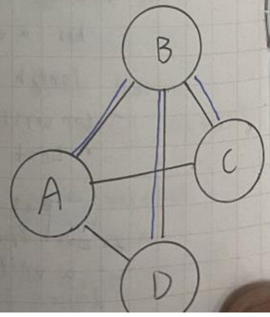
* To create a spanning tree
- Traverse the graph using either depth-first search (DFS) or breath-first search (BFS) and mark the edges that you follow
- After the traversal is completed, the graph's vertices and marked edges form a spanning tree.

DFS in iterative from (stack)

```
iterativeDFS(vertex: v)

s.createStack();
s.Push(v);                     count = 0
Mark v as visited
while(! s.isEmpty() && count < |V| -1)
{     u= s.getTop();  // at top of stack
   if ( unvisited vertex w is adjacent to u)
   {  s.Push (w);   count++;
      Mark w as visited;  // (u,w)
   }
   else  s.PoP();  // back track
```





My Questions
Problems & Difficulties needing exploration

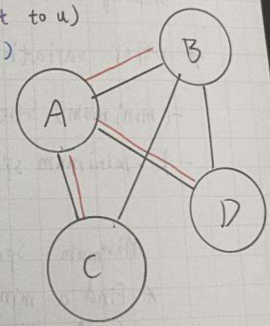
My learning weather report

BFS in iterative from (Queue)

```
iterativeBFS(vertex v)
  q.createQueue();  count = 0;
  q.enqueue(v);
  Mark v as visited;
  while(!q.isEmpty() && count < |V|-1)
  { q.dequeue(u);
    for (each unvisited vertex w adjacent to u)
    {    Mark w   as visited   //(u,w)
         q.dequeue (w);   count++;

    }

}
```



My Opinions
Thoughts, inspirations, and suggestions

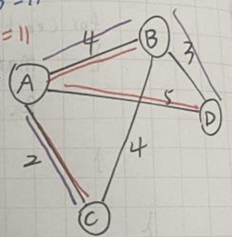
Minimum Spanning Tree: Definition

* Cost of Spanning tree
  - sum of edge weights on a spanning tree

* A minimum spanning tree of a connected undirected graph has a minimal edge-weight sum.

- A patricual graph could have several minimum spanning trees

DFS: 4+4+3=11
BFS: 4+2+5=11
MST: 4+2+3=9



* other variations
  -(minimum) Steiner tree
  - k-minimum spanning tree

Minimum Spanning Trees: Algorithms (Prim.

* Find a minimum spanning trees at any given vertex [Robert prim]

1. Find the least-cost node (v,u) from a visited vertex v to some unvisited vertex u
2. Mark u as visited
3. Add the vertex u and the edge (v,u) to the minimum spanning tree
4. Repeat the above steps untill all vertex are visited;

My Questions Minimum spanning tree's Algorithms
Problems & Difficulties needing exploration

My learning weather report

* Find a minimum spanning tree that begins at any given vertex (kruskal)

1. create a forest, where each vertex is a tree
2. Find the least-cost edge (u,v) where vertex v and vertex u are from two different trees
3. Merge the trees of vertex and vertex u, and add the edge (v,u) to the minimum spanning tree
4. Repeat the above steps until |V|-1 edges.

* Solin's Algorithm
改進 kruskal's Algorithms

1. Create a forest, where each vertex is a tree

My Opinions
Thoughts, inspirations, and suggestions

2. For each tree T, do the following steps:
2.1 Find the least-cost edge (v,u) where vertex is in T and vertex u is outide T
2.2 merge the trees of vertex v and vertex u, and add the edge (v,u) to the minimum spanning tree
3. Repeat step2 until only one tree is left

My Notes Shortest Path
Important Concepts worth keeping

Today / /

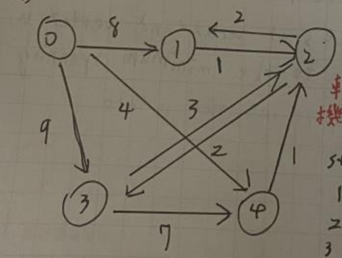
* Shortest path between two verticex in a weighted graph is the path that has the smallest sum of its edge weights

Shortest Paths: Dijkstra's Algorithm

* Problem definition
  - Find the shortest paths between a given origin and all other vertices

* Basic idea
  - A set vertexset of selected vertices
  - An array weight, where weight[v] is the cheapest weight of the shortest path from vertex 0 (origin) to vertex v that passes through only the vertices in vertex Set

(a)



只要找到一條最短路徑
ex A → B → C
則 A → B 也是最短路徑

轉換

step	v	vertex Set	weight [0]	[1]	[2]	[3]	[4]
1	-	0	0	8	∞	9	4 min
2	4	0,4	0	8	5	9	4
3	2	0,4,2	0	7	5 (4+1)	8	4
4	1	0,4,2,1	0	7	5	8	4

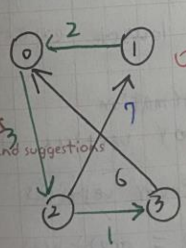
My Questions
Problems & Difficulties needing exploration

Single-Source All-Destination Shortest Paths

* Dijkstra's algorithm:

1. Initialize vertexset & weight; $V = V_0$;
2. update weight for each vertex u not in vertexSet which is adjacent to v
   weight[u] = min{ weight[u], weight[v] + edge weight(v,u) }
3. Find the shortest path from 0 to u among every path that starts from 0, passes vertices in vertexSet, and ends at a vertex not in vertexSet
   if (weight[u] is minimum) · vertexSet = vertexSet + {u};
4. Repeat step 2,3 until no more vertex can be added

All-Pairs Shortest Paths



0	0, 1	0→2→1	10
	0, 2	0→2	3
	0, 3	6→2→3	4
1	1, 0	1→0	2
	1, 2	1→0→2	5
	1, 3	1→0→2→3	6
2	2, 0	2→3→0	7
	2, 1	2→1	7
	2, 3	2→3	1
3	3, 0	3→0	6
	3, 1	3→0→2→1	16
	3, 2	3→0→2	9

My Opinions
Thoughts, inspirations, and suggestions

weather report

成功或失败的

My Notes
Important Concepts worth keeping

All-Pairs Shortest Paths: Floyd's Algorithm

Floyd-warshall algorithm

1. Intialize distance matrix D^{-1} = adjacency matrix
2. For $k = 0$ or $|V|-1$

   $D^k \leftarrow D^{k-1}$ // Add vertex k into vertexset

   For $i = 0$ to $|V|-1$

   For $j = 0$ to $|V|-1$

   $D^k[i,j] = \min\{D^{k-1}[i,j],\ D^{k-1}[i,k] + D^{k-1}[k,j]\};$

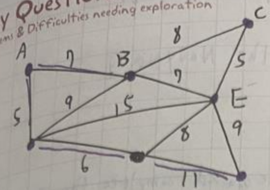
Path Finding: Best-First Search

* A* algorithm
  - Best-first Search by keeping a priority queue and terversing a path of the lowest expected total cost
  - Combine two pieces of information
    * Dijkstra's Algorithm: favor vertices close to the origin
    * Greedy best-first search: favor vertices close to the goal
  - Expected total cost: $f(v) = g(v) + h(v)$
    * $g(v)$: exact cost of the path from the origin to vertex v ✓
    * $h(v)$: heuristic estimated cost from vertex v to the goal
    * It helps efficiency if $h(v)$ is a lower bound of actual cost

lgorithm

My Questions

Problems & Difficulties needing exploration



$h(B) = 14$

$h(C) = 12 < 7+9$

$h(D) = 17 < 5+9$

$h(E) = 9$

$h(F) = 11$

$f(B) = g(B) + h(C) = 7 + 14 = 21$

$f(D) = g(B) + h(E) = 5 + 17 = 22$

$f(cap) = G(g) = 11 + 11 = 22$

My learning weather report

Graph problem

Activity-on-Edge (AOE) network

*Directed edge: activity (task) to be performed

*vertex: event to signal the completion of certain activities

My Opinions

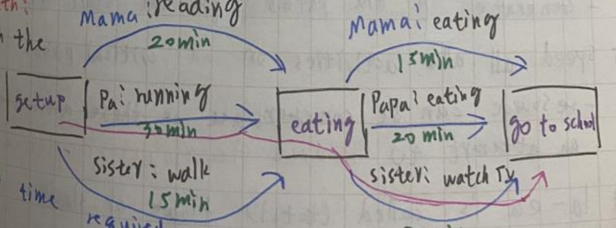
Thoughts, inspirations, and suggestions

*edge weight: the time required to perform an activity

* path length: the total time from the start to the last ev

* critical path:

- path with the longest length



- the minimum time required to complete the project

ost

origin

goal

ex √

e goal

al

My Notes Critical Path Analysis : Background

Important Concepts worth keeping

Today :

* Developed in the 1950s by the US Navy

* Input

- A list of all activities required complete the project
- The time (duration) that each activity will take to completion
- The dependencies between the activities

* output

- The longest path of planned activites to the end of the project
- The earliest time and the latest time that each activity can start and finish without making the project longer
- Determines "critical" activities (on the longest path)
- Priorize activities for the effective management and to shorten the critical path of a project

* Determine critical Paths

- Delete all non-critical activities (non zero slack)
- Generate all the paths from the start to end

* Speed up the activities on all critical paths

- resource can be concentrate on these activities in an attempt to reduce the project completion time

* la - ea is called (total) float or slack

- amount of time that a task can be delayed without causing a delay to project completion time

la - ea == 0 means a critical activity

暫停吸口氣，
靜思問自己，
關鍵10秒鐘，
Ho住十年功。

Critical Path Method: Forward Phase

My Questions
Problems & Difficulties needing exploration

My learning weather report

* Like top sort! ← Initialize: $ee[v_0] = 0$

1. Find vertex v that has no sucessor (out-degree = 0)
   Predecessor (in-degree = 0)
2. Add v to the beginning of a list
3. Remove v from the graph, as well as all edges that lead to v
4. Repeat the previous steps untill the graph is empty

2. For each immediate successor u do the following
- set $ea[x] = ee[v]$, where x is the activity on $\langle v, u \rangle$
- set $ee[u] = \max\{ee[u], ee[v] + \text{duration of } \langle v, u \rangle\}$
- Decrease the in-degree of u

3. repeat the steps untill all vertices are visited
For the vertex w that has no successor $le[w] = ee[w]$!

Critical Path Method: Backward Phase

My Opinions
Thoughts, inspirations, and suggestions

1. Find vertex u that has no successor (out-dergee = 0)
2. for each immediate predecessor v, do the following
- Set $la[x] = \{le[u] - \text{duration of } \langle v, u \rangle\}$, where x is the activity on $\langle v, u \rangle$
- set $le[v] = \min\{le[v], le[u] - \text{duration of } \langle v, u \rangle\}$
- Decrease the out derge of v
3. repeat the step untill all vertices are visited
- For the vertex w that has no predecessor, $le[w] = ee[w]$!

cipher key

慢比快還要快。

* Critical-Path analysis can be carried out with AOV network
* Free float: amoung of time that a task can be delayed without causing a delay to the earliest start of any immediately following activities
  - earliest finish time & latest finish time for each activity

Maximum Flow Problem

* we are given a flow network G with source s and sink t, and we wish to flow of maximum value from s to t
* single-source single-sink maximum flow problem
* Maximum-flow min-cut theorem

Maximum Flow Problem: Background

* A simplified model of Soviet railway traffic flow
  - Formulated by T.E Harris 1954
* Ford-Fulkerson algorithm, 1955
  - Residual graph
    * residual capacity: $c_1(u,v) = c(u,v) - f(u,v)$, $c_1(v,u) = c(v,u) - f(v,u)$
* Edmond-Karp algorithm, 1972
  - Heuristic to find augmenting path
* Residual graph
  - residual capacity: $c_1(u,v) = c(u,v) - f(u,v)$, $c_1(u,v) = c(v,u) - f(v,u)$

My Questions Ford-Fulkerson algorithm
Problems & Difficulties needing exploration

My learning weather report

1. Initialize $c_f(v,u)$ for every edge
2. Find a path P from s to t $\ni c_f(u,v) > 0 \ \forall (u,v) \in P$
3. $c_f(P) = \min\{c_f(u,v) : (u,v) \in P\}$
4. For each edge $(u,v) \in P$
   - $c_f(u,v) = c_f(u,v) - c_f(P)$
   - $c_f(v,u) = c_f(u,v) + c_f(P)$;

Edmonds-Karp algorithm

1. Initialize $c_f(v,u)$ for every edge
2. Find a path P from s to t by a heuristic
3. $c_f(P) = \min\{c_f(u,v) \mid (u,v) \in P\}$; Heuristic 1. max-capacity first
4. For each edge $(u,v) \in P$
   - $c_f(u,v) = c_f(u,v) - c_f(P)$
   - $c_f(v,u) = c_f(v,u) + c_f(P)$;

My Opinions
Thoughts, inspirations, and suggestions

Other Difficult Problem

* Eulerian circuit (Euler tour)
- Find a tour that would pass each edge exactly once and finally return to the starting vertex

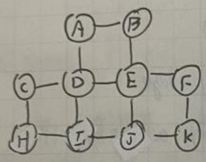
* Hamiltion circuit
- Find a tour that would vist each vertex exactly once and finally return to the staring vertex

好還可以更好
高還可以
只要一步步
密碼
cipher key

Other Difficult Problem

* Eulerian circuit (Euler tour)
- Find a tour would pass edge exactly once and finally return to the starting vertex

* DFS-based Algorithm



Not exist!

Path:
A
B
E
F
K
J {I}
E
D {C,I}
A

Traveling Salesman Problem (TSP

* Hamiltion circuit
- Find a tour would visit each vertex exactly once and finally return to the starting vertex
- Decision problem (NP-complete)

* Brute-force algorithm (暴力) 3

* Greddy algorithm 1 (不注重 answer 品質

* Branch-and-bound algorithm 2
分支 界限

My Questions Graph coloring Problem : Algorithm
Problems & Difficulties needing exploration

My learning weather report

* vertex coloring (Edge coloring)
* Sequential ordering algorithms
  - Heuristics for a specific ordering of vertices
    * No guarantee on using the least number of colors
  - welsh-Powell algorithm (greedy coloring)
    * max-degree-first

Bi-connected Graph: Algorithm

* Articulation point 關節點
  - A vertex v in G is a articulation point if: the deletion of v, together with the deletion of all edges incident to v leaves behind a graph that has two connected components (disconnected)
* Bi-connected graph

My Opinions
Thoughts, inspirations, and suggestions

  - A connected graph that has no articulation point.
* Finding the articulation points
  - Graph traversal algorithm
  - DFS-tree based algorithm

My Notes Bi-connected Graph : Definitions

Important Concepts worth keeping

Today : / /

* Finding the articulation points
  - Graph traversal algorithm
  - DFS - tree based algorithm.