Decision Maths Cribsheet

Sorting Algorithms

Bubble sort

- order a_1, a_2 (the first 2 items in list)
- order 'new' a_2,a_3
- repeat throughout list to complete 1st pass
- repeat until no changes required (final pass)

Shell Sort

- can be used as a pre-sort to improve overall efficiency
- arrange data into a 2D array with few rows
- sort each column
- arrange data again into a 2D array but with fewer columns and repeat

Quick sort

- choose pivot
- sort into 2 groups either side of pivot (smaller/greater)
- repeat process for each group and sub group until each sub group has single element

Shuttle sort

- order a_1, a_2
- order a_2, a_3 – if swap occurs order a_1, a_2
- repeat process – effectively sorting 2 nos, 3 nos, etc

this algorithm is of 'order' ½ n(n-1) - ie formula for sum of n numbers

Binary Search

applies to an ordered list

- check middle element-decide which ½ required item is in
- check middle element of selected ½
- repeat until required item is found

Packing

- first fit – put in ist available bin
- first fit decreasing – arrange in order of size (largest 1st) then apply first fit algorithm.

Graphs & Networks

Definitions

- *Graph* – a set of vertices (or nodes) and a set of edges (or arcs)
- *Edge* – an edge has a vertex at each end
- *Loop* – an edge with the same vertex at each end
- *Order (or degree)* – of a vertex is the number of edges connected to it
- *Simple Graph* – graph with no loops and maximum of one edge connecting any pair of vertices
- *Walk* – sequence of edges where end of one edge is start of next
- *Trail* – a walk with no repeated edge
- *Path* – a trail where no vertex is repeated
- *Cycle* – a path that starts and finishes at the same vertex
- *Complete* – each pair of vertices connected by an edge K_n
- *Eulerian Cycle* – traverses all edges of the graph, starts and ends at same vertex (all vertices of even order)
- *Semi-Eulerian Path* - traverses all edges of the graph, starts and ends at different vertices (2 vertices odd)
- *Hamiltonian Cycle* – passes through every vertex once only
- *Connected Graph* – graph with a path between every pair of vertices
- *Tree* – a simply connected graph with no cycles
- *Spaning Tree* – tree where vertices are all the vertices of the graph
- *Network* – graph with 'weighted' edges – ie number/value associated
- *Minimum Spanning Tree* – minimum weight tree (of a network)– also called *Minimum Connector*
- *Planar* – edges only meet at vertices (no crossing of edges)
- *Complete Graph* – a simple graph where every pair of vertices are connected by an edge K_{nn}
- *Directed Graph* – at least one edge has an associated direction (also called *a Digraph*)
- *Bipartite Graph* – vertices split into 2 sets with each edge connecting a vertex from eack set

Minimum Connectors Prim's & Kruskal's Algorithms

Prim's Algorithm

- choose a vertex (any)
- connect to vertex giving minimum weight
- connect vertex to existing tree which adds least weight
- repeat previous step until all vertices connected

Matrix form of Prim's Algorithm

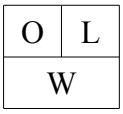
- choose any start vertex
- circle minimum weight in column
- label column as 1, delete row
- label deleted row as column 2
- circle smallest weight in column etc...

Kruskal's Algorithm

- choose minimum weight edge (any of duplicated minimum)
- find next edge of minimum weight that does not make a cycle
- repeat until all vertices connected

Shortest Path Dijkstra's Algorithm

Used to determine the shortest route between 2 vertices of a network. Each vertex is given a label box -



O – order of labelling
L – minimum distance from start vertex (label)
W – 'working' values

- label start vertex L=0, O = 1
- for each vertex connected to start vertex, enter distance from start as W value.
- enter smallest working value as label for that vertex and order = 2
- for each vertex connected to most recently labelled, add edge distance to label value to find total distance.This is working value unless lower value already found
- find vertex with smallest working value not yet labelled. Label this vertex and record order
- repeat previous 2 steps until target vertex is labelled. Value of label is the minimum distance from start vertex.
- find path by starting at target working backwards such that edge is included if distance = change in label values

Travelling Salesman Problem

Problem is to find the minimum distance travelled to visit every verte and return to start. Complex graphs are difficult to investigate completely (solve) as the number of possible routes increases rapidly with number of vertices/edges. However, it is possible to 'bound' the problem – finding (non unique) upper and lower bounds.

Upper Bound

Method 1

- twice a minimum spanning tree
- reduce by including short cuts on return route

Method 2 – nearest neighbour algorithm

- choose start vertex
- move to nearest vertex not yet visited
- repeat previous step until every vertex has been visited
- return to start

Lower Bound

- choose a vertex – delete it and all edges connected to it
- find minimum spanning tree for remaining vertices
- add lengths of 2 shortest deleted edges to minimum spanning tree

Note that different vertex choice for deletion can give 'better' ie larger lower bound

Route Inspection Problem (Chinese Postman Problem)

Problem is to find the shortest route to travel along each edge and return to the start.

If all vertices are even then the solution is trivial – the sum of all the edge distances.

For each odd vertex (there will be an even number for a connected network), one of the edges will need to be travelled twice. To minimise the route;

- pair up the odd vertices
- find the minimum distance to join up the pairs and add together
- repeat for all possible pairings and hence find minimum 'added distance.
- solution is then the sum of this minimum added distance and the sum of all the edges
- possible routes can then be found by inspection

Linear Programming

Objective is to find an optimal solution (eg max or min of a cost function) subject to a set of constraints. Process is examining the value of the cost function at each corner point of the 'feasible' region (ie all constraints satisfied). For 2 variables, solution can readily be found graphically. For more variables the same process can be performed using iterations of a 'Simplex Tableau'.

Process can best be followed by working through examples.

Simplex Tableau

A Simplex tableau comprises a 'profit' function and a set of constraints, with a 'slack' variable asociated with each constraint. Combining rows of the Tableau, called an iteration, using a 'pivot' enables the 'profit' function to be optimized. The Tableau can be interrogated after each iteration to establish values of the various parameters. As with linear programming, the process can best (only?) be understood by working through specific examples.

Bipartite Graph Matching

A maximal matching has the maximum number of edges – every vertex in one set is connected to a vertex in the other. A complete matching occurs when every vertex is connected to another. (only possible for 2 sets with same number of vertices)

Alternating path algorithm –

- starts with allowable connections between vertices of each set, and an 'initial' one to one matching.
- choose an unmatched vertex and a possible match
- find a path back deleting paths from the initial matching from this vertex (if any)
- repeat process until no further alternating paths can be found
- consider alternative start points or routes until optimal match obtained

Items to follow soon

- Network Flows
- Critical Path Analysis
- Worked examples for linear programming and Simplex Tableau
- Cross referencing to Exam board modules