Contents

enumerate{}

1 Algorithms

1. An **algorithm** is a finite step-by-step instructions carried out to solve a problem.
2. In a **flow chart**, the shape of each box tells you about its function
3. Unorderedlists can be sorted using a bubble sort or a quick sort.
4. In a **bubble sort**, you compare adjacent items in a list:
    - If they are in order, leave them.
    - If they are not in order, swap them.
    - If the list is in order when a pass is completed without any swaps.
5. In a **quick sort**, you select a pivot then split the items in two sub-lists:
    - One sub-list contains items less then the pivot
    - The other sub-list contains items greater than the pivot.
    - You can select further pivots from within each sub-list and repeat the process.
6. The three bin-packing algorithms are first-fit, first-fit-decreasing, and full-bin:
    - The **first-fit algorithm** works by considering items in the order they are given.
    - The **first-fit-decreasing** algorithm requires the itmes to be in descending order before applying the algorithm.
    - **Full-bin packing** uses inspection to select items that will combine to fill bins. Remaining items are packed using the first-fit algorithm.

7. The three bin-packing algorithms have the following advantages and disadvantages:

Type of algorithm	**Advantage**	**Disadvantage**
First-fit	Quick to implament	Not likely to lead to a good solution
First-fit-decreasing	Usually a good solution Easy to implament	May not get an optimal solution
Full-bin	Usually a good solution	Difficult to do, especially when the numbers are p awkward

8. The **order** of algorithm can be described as a function of its size. If an algorithm has order $f(n)$, then its increasing size of the problem from n to m will increase the run time of the algorithm by a factor approximately $\dfrac{f(m)}{f(n)}$

9. Ifthe size of a problem is increased by some factor k then:

   - an algorithm of linear order will take approximately k times as long to complete
   - an algorithm of quadratic order will take approximately k^2 times as long to complete
   - an algorithm of cubic order will take approximately k^3 times as long to complete.

2 Graphs and networks

1. A **graph** consists of points (called **vertices** or **nodes**) which are connected by lines (**edges** or **arcs**).

2. If a graph has a number associated with each edge (usually called its weight), then the graph is known as a **weighted graph** or **network**.

3. A **subgraph** is a graph, each of whose vertices belongs to the oridional graph and each of whose edges belongs to the origional graph. It is part of the origional graph.

4. The **degree**(or**valency** or **order**) of a vartex is the number of edges incident to it.

5. If the degree of a vertex is even, you say that it has **even degree.** If the degree of a vertex is odd it has **odd degree**.

6. A **walk** is a route through a graph along edges from one vertex to the next.

7. A **path** is a walk in which no vertex is visited more than once.

8. A **trail** is a walk in which no edge is visited more than once.

9. A **cycle** is a walk in which the end vertex is the same as the start vertex and no other vertex is visited more than once.

10. A **Hamiltonian cycle** is a cycle that incles every vertex.
11. Two vertices are **connected** if there is a path between them. A graph is connected if all its vertices are connected.
12. A **loop** is an edge that starts and finishes at the same vertex.
13. A **simple graph** is one which there are no loops and there is most one edge connecting any pair of vertices.
14. If the edges of a graph have a direction associated with them they are known as **directed edges** and the graph is known as a **directed graph** (or **digraph**).
15. In any undirected graph, the sum of the degrees of the **vertices** is equal to $2x$ the number of edges. As a consequence, the number of odd nodes must be even. This is called **Euler's handshaking lemma**.
16. A **tree** is a connected graph with no cycles.
17. A **spanning tree** of a graph is a subgraph which includes all the vertices of all origional graph and alsa a tree.
18. A **complete graph** is a graph in which every vertex is directly connected by a single edgeto each of the other vertices.
19. **Isomorphic graphs** are which show the same information buy may be drwawn differntly.
20. Each entry in an **adjacency matrix** describes the number of arcs joining the corresponding vertices.
21. In a **distance matrix**, the entries represent the weight of eacha arc, not the number of arcs.
22. A **planar graph** is one that can be drawn in a plane sucht that no two edges meet except at a vertex.
23. The **planarity algorithm** may by applied to any graph that consists a Hamiltonian cycle. It provides a method of redrawing a graph in such a way it becomes whether or not its planar.

3 Algorithms on graphs

1. A **minimum spanning tree** (MST) is a spanning tree such that the total length of its arcs (edges) is as small as possible.
2. **Kruskal's algorithm** can be used to find a minimum spanning tree.
    - Sort the arcs into ascending order of weight and use the arc of least weight to start the tree. Then add arcs in order of ascending weight, unless an arc would form a cycle, in which case reject it.
3. **Prim's algorithm** can be used to find a minimum spanning tree.

- Choose any vertex to start the tree. Then select an arc of least weight that joins a vertex already in the tree to a vertex not yet in the tree. Repeat until all vertices are connected.
- Prim's algorithm can be applied to a **distance matrix**.

- Choose any vertex to start the tree. Delete any row in the matrix for the chosen vertex and number the column in the matrix for the chosen vertex. Ring the lowest undeleted entry in the numbered columns, which becomes the next arc. Repeat this until all rows are deleted.

4. **Dijkstra's algorithm** can be used to find the shortest parth between two vertices in a network.

   - Label the start vertex with the value O.
   - Record a working value at every vertex, Y, that is directly connected to the vertex that has just recieved its final label, X:

     final label at X + weight of arc XY
   - Select the smallest working value. This is the final value for the vertex.
   - Repeat until the destination vertex recieves its final label.
   - Find the shortest path by tacking back from destination to start. Include an arc AB on the route if B is already on route and

     final label B - final label A = weight of arc AB

   Each vertex is replaced by a box

Vertex	Order of labelling	Final label
Working values		

5. Dijstra's algorithm finds the shortestr oute between the start vertex and each intermediate vertex completed on the way to the destination vertex.

6. It is possible to use Dijkstra's algorithm on networks with directed arcs, such as as route with one-way streets.

7. **Floyd's algorithm** can be used to find the shortest path between every pair of vertices in a network.

   - Floyd's algorithm applied to a network with n vertices produces two tables as a result of n iterations. One table shows the shortest distancesa and the other constrains informationabout the corresponding paths taken.

4 Route inspection

1. An **Euleian graph** or network is one which contains a trail that includes every edge and starts and finishes at the same vertex. This trail is called a Eulerian circuit. Any connected graph whose vertices are all even are Eulerian.

2. A **semi-Eulerian graph** or network is one which contains a trail that includes every edge but starts and finishes at different vertices. Any connected graph with exactly two odd vertices is semi-Eulerrian.

3. The **route inspection algorithm** can be used to find the shortest route in a network that traverses very arc at least once and returns to its starting point.

   - If all the vertices in the network have even degree, then the length of the hosrtest route will be equal to the total weight of the network.
   - If a network has exactly two odd vertices, then the length of the shortest route will be equal to the total weight of the network, plus the length of the shortest path between the two odd vertices.
   - If the network has more than two odd vertices, then you need to consider all the possible paring of the odd vertices.

4. Here is the route inspectino algorithm.

   - Identify any vertices with odd degree
   - Consider alll possible pairings of these vertices
   - Select the complete pairing that has the least sum
   - Add a repeat of the arcs indicated by this paring to the network

5 The travelling salesment problem

1. A **walk** in a network is a finite sequence of edges such that the end vertex of one edge is the start vertex of the next.

A walk visits every vertex, returning to its starting verteex, is called a **tour**.

2. The are two variations of the travelling salesman problem. In the **classical problem**, each vertex must be visited exactly once before returning to the start. In the **practical problem**, each vertex must be visited at least once before returning to the start.

3. The **triangle inequality** states

   the longest side of any triangle $\leq$ the sum of the two shorter sides.

4. You can use a **minimum spanning tree method** to find an upper bound for the practical travelling salesman problem.

   - Find the minimum spanning tree for the network (using Prim's algorithm or Krunkal's algorithm).
   - An **initial upper bound** for the practical travelling salesman problem is found by finding the weight of the minimum spanning tree for the network and doubling it.

- You can improve the initial upper bound by looking for **shortcuts**. Aim to make the upper bound as low as possible to reduce the interval in which the optimal solution is contained.

5. You can use a **minimum spanning tree method** to find a lower bound from a table of least distances for the classical problem.
   - Remove each vertex in turn, together with its arcs.
   - Find the residual minimum spanning tree (RMST) and its length.
   - Add the RMST the 'cost' of reconnecting the delted vertex **by the two shorteset, distinct, arcs** and note the totals.
   - The greatest of these totals is used for the lower bound.
   - Make the lower bound as high as possible to reduce the interval in which the optimal solution is contained.
   - You have found an optimal solution if the lower bound gives a Hamiltonian cycle, or the lower bound has the same value as the upper bound.
6. You can use the **nearest neighbour algorithm** to find the upper bound.
   - Select each vertex in turn as a starting point.
   - Go to the nearest vertex which has not yet been visited.
   - Repeat step **2** until all vertices have been visited and then return to the start vertex using the shortest route.
   - Once all vertices have been used as the starting vertex, select the tour with the smallest length as the upper bound.

6 Linear programming

1. To formulate a problem as **linear programming** problem:
   - Define the **desicion variable** ($x, y, z,$ etc,).
   - State the **objective** (maximise or minimise, together with an algebraic expression called the **objective function**).
   - Write the **constraints** is inequalities.
2. The region of a graph that satisfies all the constraints of a linear problem is called the **feasable region**.
3. To solve a linear programming problem, you need to find the point in the feasible region which maximises or minimises the objective function.
4.
   - For a **miximum point**, look for the last point covered by an objective line as it leaves the feasible region.
   - For a **minimum point**, look for the first point covered by an objective line as it leaves the feasible region.
5. To find an optimal point using the **vertex method**:

- First find the coordinates of each vertex of the feasable region.
- Evaluate the objective function at each of these points.
- Select the vertex that gives the optimal valueof the objective function.

6. If a linear programming problem required integar solutions, you need to consider points with integar coordinates near the optimal vertex.

7 This simplex algorithm

1. To fromulate a **linear programming** problem:
   - define your decision variables
   - write down the objective function
   - write down the constraints
2. Inequalities can be transformed into **equations** using **slack variables** (so called because they represent the amount of slack between an actual quantity and the maximum possible vlaue of that quantitiy).
3. The **simplex method** allows you to:
   - determine if a particular vertex, on the edge of the fesible region is optimal
   - decide which adjacent vertex you should move to in order to increase the value of the objective function.
4. **Slack variales** are essential when using the simplex algorithm.
5. The simplex method always starts from a basic fesible region, at the origin, and then progresses which each iteration to an adjacent point whithin the feasible region until the optimal solution is found.
6. To use a **simplex tableau** to solve a maximising linear programming problem, where the constraints are given as equalities:
   - Draw the tabluau; you need a basic variable column on the left, one column for each variable (including the slack variables) and a va;ie column. You need one row for each constraint and the bottom row for the objective function.
   - Create the initial tableau: enter the coefficietns of the variables in the appropriate column and row.
   - Look along the objective row for the most negative entry; this indicates the pivot column.
   - Calculate θ, for each of the constraint rows, where
     $$\theta = \frac{\text{the term in the value column}}{\text{the term in the pivot column}}$$
   - Select the row with the smallest, positive θ value to become the pivot row.

- The element in the pivot row and pivot column is the pivot.
- Devide all of the elements in the pivot row by the pivot, and change the basic variable at the start of the row to the variable at the top of the pivol column.
- Use the pivot row to eliminate the pivot's variable from the order-rows: this means that the pivot column now contains one 1 and zeros.
- Repeat bullets 3 to 8 until there are no numbers in the objective row.
- The tableau is now optimal and the non-zero values can be read off using the basic variable column and the value column.

7. The steps for solving a minimising linear programming problem are identical to those given above apart from:

    - First, define a new objective function that is the negative of the origional objective function.
    - AFter you have maximised this new objective function, write your solution as the negative of this value, which will maximise the origional objective function.

8. When integar solutions are needed, test points arount the optimal solution to find a set of points which fit the constraints and give a maximum for the objective function.

9. The **two-stage symplex method** for problems that include $\geq$ constraints:

    - Use **slack**, **surplus** and **artifitial** variables, as necessary, to write all constraints as equations.
    - Define a new objective function to minimise the sum of all artifitial variables.
    - Use the simplex method to solve this problem.
    - If the minimus sum of artifitial values is 0 then the solution found is a basic fesible solution of the origional problem, which is a starting point to the second stage. Use the simplex method agian to solve this problem.
    - If the minimus sum of the artifitial varaiables is not 0 then the origional problem has no feasible solution.

10. Linear programming problems that include *leq* may also be used using the **Big-M method** instead of the two-stage simplex method. The Big-M method uses an arbitrary large number, M, in the objective function, Its purpose is to drive the artifitial variables towards 0.

11. The **Big-M method** using the following steps:

    - Introduce a slack variable for each constraint of the form $\leq$
    - Introduce a surplus variable for each constraint of the form $\geq$

- For each artifitial variable a_1, a_2, a_3 ... subtract $M(a_1 + a_2 + a_3 \ldots)$ from the objective function, where M is an arbitrary large number.
- Eliminate artifitial variables from your objective function so that the variables remaining in your ojbective function are non-basic variables.
- Formulate an initial tableau, and apply the simplex algorithm in the normal way.

8 Critical path analysis

1. A **precedence table**, or **dependence table**, is a table that shows which **activities** must be completed before others are started.
2. In an **activity arc network**, the activities are represented by **arcs** and the commpletion of those activites, known as **events**, are shown as **nodes**.
   - Each arc is labelled with an activity letter.
   - The beginning and end of an activity are shown at the end of the arc and an arrow is used to define the direction. The convention is to use straight lines for arcs.
   - The nodes are numbered starting from 0 for the first node, which is called the **source node**.
   - Sometimes the source node is lablled 1 instead of 0.
   - Number each node as it is added to the network
   - The final node is called the **sink node**.
3. A **dummy activity** has no time or cost. Its sole puropose is to show the dependencies between activities.
4. Every activity must by **uniquely identified** in terms of its events. The requires that there can be **at most one activity** between any two events.
5. The length of time of an activity takes to complete is known as its duration. You can add weights to the arcs in an activity network to represent these times.
6. - The **early event time** is the earlies time of arrival at the event allowing for the completion of all preceding activities.
   - The **late event time** is the latest time that the event can be left without extending the time needed for the project.
   - The early event times are calculated starting from 0 at the source node and working towards the sink node. This is called a **forward pass** or **forward scan**.
   - The late event times are calculated starting from the sink node and working backwards towards the source node. This is called a **backwards pass** or **backward scan**.

7. - An activity is described as a **critical activity** if any increase in its duration results in a corresponding increase in the duration of the whole project.
   - A path from the source node to the sink node which entirely flows critical activities is called a **critical path**. A critical path is the longest path contained in the network
   - At each node (vertex) on a critical path **the early event time is equal to the late event time**.

8. An activitiy connecting two critical events isn't necessarily a critical activity.

9. - The **total float** of an activity is the amount of time that its start my be delayed without affecting the duration of the project.

     Total float = latest finish time - duration - earliest start time
   - The total float of any critical activity is 0.

10. A **Grantt** (cascade) **chart** provides a graphical way to represent the range of possible start and finish times for all activites on a single diagram.

11. You need to be able to consider the number of **workers** needed to complete a project. You will be told the number of workers that are needed for an activity.
    - No worker can carry out more than one activity simultaneosly.
    - Once a worker, or workers, have started an activity, they must complete it.
    - Once a worker, or workers, have finnished an activity, they immediately become available to start another activity.

12. The process of adjusting the stsart and finish times of the activities in order to take into account the constraints on resources is called **resource levelling**.

13. When you are scheduling a project:
    - you should allways use the first available worker
    - if there is a choice of activities for a worker, assign the one with the lowest value for its late time.

14. The lower bound for the number of activities of workers to complete a project within its critical time is the smallest integar greater than or equal to:

$$\frac{\text{sum of all of the activity times}}{\text{critical time of the project}}$$