

Algorithms II Cheat-Sheet

Notation

$A \in [10] \equiv A \in [1..10]$
 $\{a, b, c\}$ is a set of vertices
 $G\{a, b, c\}$ is a graph
 \vec{x} just represents a vector x

Big O

Notation	Intuitive meaning	Analogue
$f(n) \in O(g(n))$	f grows at most as fast as g	\leq
$f(n) \in \Omega(g(n))$	f grows at least as fast as g	\geq
$f(n) \in \Theta(g(n))$	f at the same rate as g	$=$
$f(n) \in o(g(n))$	f grows strictly less fast than g	$<$
$f(n) \in \omega(g(n))$	f grows strictly faster than g	$>$

Notation	Formal definition
$f(n) \in O(g(n))$	$\exists C, n_0: \forall n \geq n_0: f(n) \leq C \cdot g(n)$
$f(n) \in \Omega(g(n))$	$\exists c, n_0: \forall n \geq n_0: f(n) \geq c \cdot g(n)$
$f(n) \in \Theta(g(n))$	$\exists C, c, n_0: \forall n \geq n_0: c \cdot g(n) \leq f(n) \leq C \cdot g(n)$
$f(n) \in o(g(n))$	$\forall C: \exists n_0: \forall n \geq n_0: f(n) \leq C \cdot g(n)$
$f(n) \in \omega(g(n))$	$\forall c: \exists n_0: \forall n \geq n_0: f(n) \geq c \cdot g(n)$

Interval Scheduling

A **request** is a pair of integers (s, f) with $0 \leq s \leq f$.
 We call s the **start time** and f the **finish time**.

A set A of requests is **compatible** if for all distinct $(s, f), (s', f') \in A$, either $s' \geq f$ or $s \geq f'$ — that is, the requests' time intervals don't overlap.

Interval Scheduling Problem

Input: An array \mathcal{R} of n requests $(s_1, f_1), \dots, (s_n, f_n)$.

Desired Output: A compatible subset of \mathcal{R} of maximum possible size.

Algorithm: GREEDYSCHEDULE

Input: An array \mathcal{R} of n requests.

Output: A maximum compatible subset of \mathcal{R} .

```

1 begin
2   Sort  $\mathcal{R}$ 's entries so that  $\mathcal{R} \leftarrow [(s_1, f_1), \dots, (s_n, f_n)]$  where  $f_1 \leq \dots \leq f_n$ .
3   Initialise  $A \leftarrow []$ ,  $\text{lastf} \leftarrow 0$ .
4   foreach  $i \in \{1, \dots, n\}$  do
5     if  $s_i \geq \text{lastf}$  then
6       Append  $(s_i, f_i)$  to  $A$  and update  $\text{lastf} \leftarrow f_i$ .
7   Return  $A$ .
```

Complexity:

Step 2 takes $O(n \log n)$

Steps 3–6 all take $O(1)$ time and are executed at most n times.

$\therefore \text{totalrunningtime} = O(n \log n) + O(n)O(1) = O(n \log n)$.

Interval Scheduling

Formal GreedySchedule:

$A^+ := \text{argmin} \{f : (s, f) \in R, A \cup \{(s, f)\} \text{ is compatible}\}$ for all $A \subseteq R$,

$A_0 := \emptyset, \quad A_{i+1} := A_i \cup \{A_i^+\}$

$t := \max\{i: A_i \text{ is defined}\}$

Interval Scheduling Proofs

Lemma: Greedy Schedule always outputs A_t

Proof: By induction form the following loop invariant. At the start of the i 'th iteration of 4-7:

- A is equal to $A_t \cap \{(s_1, f_1), \dots, (s_{i-1}, f_{i-1})\}$
- lastf is equal to the latest finish time of any request in A (or 0 if $A = []$)

Lemma: A_t is a compatible set

Proof: Instant by induction; A_0 is compatible, and if A_i is compatible then so is $A_{i+1} = A_i \cup A_i^+$ by the definition of A_i^+

Lemma: A_t is a maximum compatible subset of the Array R (look in pseudocode)

Proof:

Base case for $i = 1$: A_0^+ is the fastest finishing request in R by definition

Inductive step: Suppose A_i finishes faster than B_i .

Let B_i^+ be the $(i+1)$ 'st fastest-finishing element of B . Since A_i finishes faster than B_i , $A_i \cup \{B_i^+\}$ is compatible. Hence by definition, A_i^+ exists and finishes no later than B_i^+

Theorem: GreedySchedule outputs A_t , which is a maximum compatible set.

Proof: putting all of the above proofs together, we prove the theorem.

Graph Theory

Graph: $G = (V, E)$

Edge: $E = E(G)$ is a set of edges contained in $\{\{u, v\} : u, v \in V, u \neq v\}$

Vertex: $V = V(G)$ is a set of vertices

Subgraph: $H = (V_H, E_H)$ of G is a graph with $V_H \subseteq V$ and $E_H \subseteq E$

Induced Subgraph: is a subgraph if $E_H = \{e \in E : e \subseteq V_H\}$

Component: H of G is a maximal connected induced subgraph of G .

Degree: $d(v) = |N(v)|$

Neighbourhood: $N(v) = \{w \in V : \{v, w\} \in E\}$

Walk: sequence of vertices $v_0 \dots v_k$ such that $\{v_i, v_{i+1}\} \in E$ for all $i \leq k-1$

Length: the value of k (see above walk definition)

Euler Walk: a walk that contains every edge in G exactly once.

Isomorphism: two graphs are isomorphic if there is a bijection $f: V_1 \rightarrow V_2$ such that $\{f(u), f(v)\} \in E_2$ if and only if $\{u, v\} \in E_1$

Path: is a walk in which no vertices repeat

Connected: A graph is connected if any two vertices are joined by a path

Digraph: is a pair $G = (V, E)$, V is a set of vertices and E is a set of edges contained in $\{(u, v) : u, v \in V, u \neq v\}$

Strongly connected: G is .. if for all $u, v \in V$, there is a path from u to v and a path from v to u .

Weakly connected:

In-Neighbourhood: $N^-(v) = \{u \in V(G) : (u, v) \in E(G)\}$

Out-Neighbourhood: $N^+(v) = \{w \in V(G) : (v, w) \in E(G)\}$

Cycle: is a walk $W = w_0 \dots w_k$ with $w_0 = w_k$ and $k \geq 3$, in which every vertex appears at most once except for w_0 and w_k (which appear twice)

Hamilton cycle: is a cycle containing every vertex in the graph

k-regular: a graph is .. if every vertex has degree k

Bijection:

Graph Theory

Theorem: If G has an Euler walk, then either:

- every vertex of G has even degree; or
- all but two vertices v_0 and v_k have even degree, and any euler walk must have v_0 and v_k as endpoints

Theorem: let $G = (V, E)$ be a digraph with no isolated vertices, and let $U, v \in V$. Then G has an Euler walk from u to v if and only if G is weakly connected and either:

- $u = v$ and every vertex of G has equal in- and out-degrees; order
- $u \neq v, d^+(u) = d^-(u) + 1, d^-(v) = d^+(v) + 1$ and every other vertex of G has equal in- and out-degrees

Dirac's Theorem: Let $n \geq 3$. Then any n -vertex graph G with minimum degree at least $\frac{n}{2}$ has a Hamilton cycle.

Handshake lemma: For any graph

$G = (V, E), \sum_{v \in V} d(v) = 2|E|$

Proof: All edges contain two vertices, and each vertex v is in $d(v)$ edges. Count the number of vertex-edge pairs: Let $X = \{(v, e) \in V \times E : v \in e\}$. Then $|X| = 2|E|$ and $|X| = \sum_{v \in V} d(v)$, so we're done.

Directed Handshake lemma: For any graph

$G = (V, E), \sum_{v \in V} d^+(v) = \sum_{v \in V} d^-(v) = 2|E|$

Proof: TODO. Instead of counting vertex-edge pairs, we count tail-edge pairs. Each edge has one tail so $|X| = |E|$

Trees

Forest: a graph with no cycles

Tree: a forest that is connected

Root: for $T = (V, E)$. Root $r \in V$ as follows. For all vertices $v \neq r$, let P_v be the unique path from r to v . Then direct each P_v from r to v .

Leaf: is a degree-1 vertex. Root cannot be a leaf.

Ancestor: u is an .. of v if u is on P_v

Parent: u is the .. of v if $u \in N^-(v)$

level: first .. L_0 of T is r , and $L_{i+1} = N^+(L_i)$.

depth: of T is $\max\{i: L_i \neq \emptyset\}$. Root doesn't count

Lemma 1: If $T = (V, E)$ is a tree, then any pair of vertices $u, v \in V$ is joined by a unique path uTv in T .

Lemma 2: Any n -vertex tree has $n-1$ edges

Lemma 3: Any n -vertex tree $T = (V, E)$ with $n \geq 2$ has at least 2 leaves

Tree Properties:

A: T is connected and has no cycles

B: T has $n-1$ edges and is connected

C: T has $n-1$ edges and has no cycles

D: T has a unique path between any pair of vertices

$A \implies B, C, D$

$A \iff B, C, D.$

Search and Dijkstra

Adjacency Matrix:

Storing: $\Theta(|V|^2)$ space

Adjacency query: $\Theta(1)$ time

Neighbourhood query: $\Theta(|V|)$ time

Adjacency List:

s $\rightarrow b, a$

a $\rightarrow s, c$

Storing: $\Theta(|V| + |E|)$ space

Adjacency query: $\Theta(d^+(u))$ time

Neighbourhood query: $\Theta(d^+(u))$ time

Linear Programming

Feasible: We say $\vec{x} \in \mathbb{R}^n$ is a feasible solution if $\vec{x} \geq \vec{0}$ and $A\vec{x} \leq \vec{b}$.

Optimal: We say \vec{x} is an optimal solution if $f(\vec{y}) \leq f(\vec{x})$ for all feasible $y \in \mathbb{R}^n$

Polytope: is a geometric object with flat sides

Corollary: There will always be an optimal solution

Non-Standard Form:

$-4x + 5y - z \rightarrow \max$ subject to

$x + y + z \leq 5;$

$x + y + z \geq 5;$

$x + 2y \geq 2;$

$x, z \geq 0.$

Standard Form:

$-4x + 5(y_1 - y_2) - z \rightarrow \max$ subject to

$x + (y_1 - y_2) + z \leq 5;$

$-x - (y_1 - y_2) - z \leq -5;$

$-x - 2(y_1 - y_2) \leq -2;$

$x, y_1, y_2, z \geq 0.$

Matrix Form:

$-4x + 5y_1 - 5y_2 - z \rightarrow \max$ subject to

$$\begin{pmatrix} 1 & 1 & -1 & 1 \\ -1 & -1 & 1 & -1 \\ -1 & -2 & 2 & 0 \end{pmatrix} \begin{pmatrix} x \\ y_1 \\ y_2 \\ z \end{pmatrix} \leq \begin{pmatrix} 5 \\ -5 \\ -2 \end{pmatrix};$$

$x, y_1, y_2, x \geq 0.$

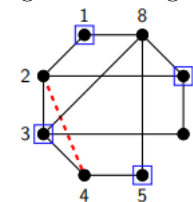
Simplex Method: Search greedily for a vertex of the feasible polytope which maximises the objective function

Worst case: hypercube which has $\Omega(2^n)$ vertices.

In practice it only need $\Theta(n)$ steps

Vertex Cover: in a graph G , is a set $X \subseteq V$ such that every edge in E has at least one vertex in X

We can express finding a minimum vertex cover as solving a linear program in which the solutions must be integers: an integer linear program.



$X = \{1, 3, 5, 7\}$ is **not** a vertex cover.

Here we have $x_1 = x_3 = x_5 = x_7 = 1$ and $x_0 = x_2 = x_4 = x_6 = 0$.

The uncovered edge $\{2, 4\}$ corresponds to the constraint $x_2 + x_4 \geq 1$, which is violated.

$$\begin{aligned} \sum_v x_v &\rightarrow \min \text{ subject to} \\ x_u + x_v &\geq 1 \text{ for all } \{u, v\} \in E; \\ x_v &\leq 1 \text{ for all } v \in V; \\ x_v &\geq 0 \text{ for all } v \in V; \\ x_v &\in \mathbb{N} \text{ for all } v \in V. \end{aligned}$$

Flow Networks

Flow Network: consists of a directed graph $G = (V, E)$, a **capacity function** $c : E \rightarrow \mathbb{N}$, a **source** vertex $s \in V$ with $N^-(s) = \emptyset$, and a **sink** vertex $t \in V$ with $N^+(t) = \emptyset$

Flow: is a function in (G, c, s, t) $f : E \rightarrow \mathbb{R}$ with properties:

- No edge has more flow than capacity;
formally, for all $e \in E$, $0 \leq f(e) \leq c(e)$
- Flow is conserved at vertices; flow in = flow out

Maximum Flow: a flow f maximising the value of the flow, $v(f)$

Cut: is any pair of disjoint edges $A, B \subseteq V$ with $A \cup B = V$, $s \in A$ and $t \in B$.

Lemma 1: For all sets $X \subseteq V \setminus \{s, t\}$, we have $f^+(X) = f^-(X)$. So flow is conserved in sets/cuts as well as vertices

Proof: By summing conservation of flow over all $v \in X$:

$$\sum_{v \in X} \sum_{u \in N^-(v)} f(u, v) = \sum_{v \in X} \sum_{w \in N^+(v)} f(v, w).$$

For all $e \subseteq X$, $f(e)$ appears once on each side; after cancelling those terms we're left with $f^+(X) = f^-(X)$.

Lemma 2: For all cuts (A, B) , $f^+(A) - f^-(A) = f^-(B) - f^+(B)$.

Proof: We have shown that $v(f) = f^+(A) - f^-(A)$ because A and B are disjoint and $A \cup B = V$.

Lemma 3: $\text{Push}(G, c, s, t, f, P)$ returns a new flow f' , with value $v(f') = v(f) + C$ in $O(|V(G)|)$ time

Ford-Fulkerson:

Input : A (weakly connected) flow network (G, c, s, t) .

Output : A flow f with no augmenting paths.

```

1 begin
2   Construct the flow  $f$  with  $f(e) = 0$  for all  $e \in E(G)$ .
3   Construct the residual graph  $G_f$ .
4   while  $G_f$  contains a path  $P$  from  $s$  to  $t$  do
5     Find  $P$  using depth-first (or breadth-first) search.
6     Update  $f \leftarrow \text{Push}(G, c, s, t, f, P)$ .
7     Update  $G_f$  on the edges of  $P$ .
8   Return  $f$ .
```

Complexity: Every step takes $O(|E|)$ time or $O(|V|)$ time, and since G is weakly connected we have $|V| = O(|E|)$. So the running time is $O(v(f^*)|E|)$.

Flow Networks

Residual graph: G_f of (G, c, s, t) on $V(G)$ as follows:

- if flow $<$ capacity: then forward edge with value capacity-flow
- if flow $>$ 0: add backward edge with value flow

Residual capacity of edge: $\max\{\text{capacity} - \text{flow}, \text{backward edge flow}\}$

Residual capacity of network: minimum residual capacity of it's edges

Augmenting Path:

