Algorithms II Cheat-Sheet

Tips

Apply an algorithm you know in a clever way, don't write a new algorithm.

Notation

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

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

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 $(s_1, f_1), \ldots, (s_n, f_n)$.
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 R's entries so that R ← [(s_1, f_1), ..., (s_n, f_n)] where f_1 ≤ ··· ≤ f_n.
3     Initialise A ← [], lastf ← 0.
4     foreach i ∈ {1, ..., n} do
5         if s_i ≥ lastf then
6             Append (s_i, f_i) to A and update lastf ← 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.
$\therefore\ totalrunningtime = O(nlogn) + O(n)O(1) = O(nlogn)$.

Interval Scheduling

Formal GreedySchedule
$A^+ := \text{argmin}\ \{f\ :\ (s,f)\ \in\ R, A \cup \{(s,f)\}$ is compatible$\}$ for all $A \subseteq R$,
$A_0 := \emptyset$, $\quad A_{i+1} := A_i \cup \{A_i^+\}$
t := max{i: A_i is defined}

Interval Scheduling Proofs

Lemma: Greedy Schedule always outupts 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 $A_t \cap \{(s_1, f1), ..., (s_{i-1}, f_{i-1})\}$
- 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 $A_{i+1} = A_i \cup A^+$ 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 fastetst-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

Definitions:
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 $v_0 ... v_k$ 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: $V_1 \to V_2$ 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 $W = w_0 ... w_k$ 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 eachvertex v is in d(v) edges. Count the number of verted-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

Definitions:

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 i 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 ∈ 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

$A \implies B, C, D$ $\qquad$ $A \impliedby B, C, D.$