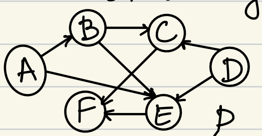
## 0.1 Topological Orderings

**Def.** Directed Acyclic Graph (DAG) - directed graph with no directed cycles

Precedence constraints: edge $(v_i, v_j)$ means $v_i$ must happen before $v_j$

**Def.** Topological ordering of directed graph $G = (V, E)$ – ordering of nodes $(V)$ $v_1, v_2, \dots, v_n$ s.t. for every edge $(v_i, v_j)$ $i < j$




constrains: before outgoing edges, after incoming edges

$A < B, E$ $\quad B, D < C < F$ $\quad A, B, D < E < F$

$A < B < C$ $\quad D < C, E$ $\quad C, E < F$

D

$\Rightarrow A \to B \to E \to C \to F$ $\quad \Rightarrow \{A, B, D, E, C, F\}, \{A, D, B, E, C, F\}, \{D, A, B, E, C, F\}$

```
function TopologicalOrdering(G)
    if |V| = 0 then
        return empty list
    find a node v with no incoming edges
    remainder <- TopologicalOrdering(G - {v})
    return v + remainder
```

$O(n+m)$, $|V| = n$, $|E| = m$

**Lemma** $G$ – DAG $\to$ $G$ has a node with no incoming edges

<u>Proof by contradiction</u>

$G$ – DAG

Suppose $\forall$ node $\to$ $\exists$ incoming edge to node.

Pick a node and follow $(|V|)$ edges backwards.

Must have visited a node twice. $\Rightarrow$ $G$ has a cycle. $\Rightarrow$ Contradiction.

**Lemma** $G$ – DAG $\to$ $G$ has topological ordering

<u>Proof by induction</u>

Base case: $|V| = 1$, holds because $G$ is a topological ordering.

Induction Hypothesis: $G$ - DAG, $|V| \le k$ $\to$ $G$ has topological ordering.

Induction step:

Given DAG $G$, $|V| = k+1$.

Find node $v$ with no incoming edges ($\exists v$ by previous lemma).

$G - \{v\}$ is DAG, removing $v$ cannot create cycles.

By IH, $G - \{v\}$ has $|V| = k$ $\to$ has topological ordering $T$.

$\Rightarrow \{v, T\}$ is valid topological order of $G$, $v$ has no incoming edges.

0.2 Time Complexity

**Def.** $T(n)$ is $O(f(n))$ iff $\exists c > 0 \in \mathbb{R},\ n > 0 \in \mathbb{N}$ s.t. $\forall n \geq n_0$: $T(n) \leq c f(n)$.

**Def.** $T(n)$ is $\Omega(f(n))$ iff $\exists c > 0 \in \mathbb{R},\ n > 0 \in \mathbb{N}$ s.t. $\forall n \geq n_0$: $T(n) \geq c \cdot f(n)$.

**Def.** $T(n)$ is $\Theta(f(n))$ iff $T(n)$ is $O(f(n))$ and $T(n)$ is $\Omega(f(n))$.

$\forall d > 0 \in \mathbb{R}$, $\log n$ is $O(n^d)$

$O(\log_a(n)) = O(\log_b(n))$ for any constant $a, b > 0$ because $\log_a(n) = \frac{\log_b(n)}{\log_b(a)}$ (constant ratio)

1.1 Cycling Trip

Problem: Selecting breakpoints

route of length $L$, camp at points with distances $b_1, b_2, \ldots, b_n$ from start point

at most $C$ kilometers per day, reach end in as few days as possible

Solution: Camp at last possible campsite each day.

```
sort breakpoints in descending order (0 = b1 <= b2 <= … <= bn = L)
S <- {0}
x <- 0
while x != L do
    p - largest integer s.t. bp <= x + C
    if bp = x then
        return No solution
    x <- bp
    S <- S U {p}
return S
```

$O(n \log n)$

1.2. Proof by Contradiction

**Th** Greedy is optimal

Assume greedy is not optimal.

Let $0 = g_1 < g_2 < \ldots < g_p = L$ denote campsites chosen by greedy.

Let $0 = f_1 < f_2 < \ldots < f_q = L$ denote campsites in optimal solution where $f_1 = g_1, f_2 = g_2, \ldots, f_r = g_r$

for the largest possible value of $r$, but $f_{r+1} \neq g_{r+1}$

Note that $g_{r+1} \geq f_{r+1}$ by greedy choice of algorithm, so $g_{r+1} > f_{r+1}$

Since $g_{r+1} - g_r \leq C$ and $f_{r+2} - g_{r+1} < f_{r+2} - f_{r+1} \leq C \Rightarrow$ replacing $f_{r+1}$ with $g_{r+1}$ maintains optimality.

## 1.3 Proof by Induction

Lemma $\forall r$, $f_r \le g_r$, $g$ - Greedy campsites, $f$ - optimal solution (Greedy stays ahead)

Base case $(r=1)$: $f_1 \le g_1$

IH: $f_k \le g_k$

Induction step:

To prove: $f_{k+1} \le g_{k+1}$

By IH, $f_k \le g_k$

$g_{k+1}$ - within reach $(\le C)$ but as far as possible from $g_k$.

$f_{k+1}$ - not further than the farthest reachable from $f_k$, but by IH, definitely reachable from $g_k$.

$\Rightarrow f_{k+1} \le g_{k+1}$

Th Greedy is optimal

Proof by contradiction

$g \to k$ campsites, $f \to m$ campsites.

Suppose greedy is not optimal $\Rightarrow k > m$.

By Lemma, $f_k \le g_k$.

$\Rightarrow m \ge k$

## 1.4. Interval Scheduling

Job (activity) $j$: starts at $s_j$ and finishes at $f_j$.

Two jobs are compatible if no overlap.

Find maximum number of compatible intervals (jobs).

Solution: Consider jobs in ascending order of finishing time $f_i$.

```
sort jobs by ascending finish time (f1 <= f2 <= … <= fn)
A <- {}
for k <- 1 to n do
    if job k compatible with A then
        A <- A U {k}
return A
```

$O(n \log n)$

1.5. Greedy Stays Ahead

**Lemma** $\forall r \le k: f(i_r) \le f(j_r)$, i - Greedy schedule, j - optimal one (Greedy stays ahead)

Proof by induction

Base case ($r=1$): by order of jobs $f(i_1) \le f(j_1)$

IH: $f(i_r) \le f(j_r)$

Induction step:

To prove: $f(i_{r+1}) \le f(j_{r+1})$

$f(j_r) \le s(j_{r+1})$ no overlap.

$f(i_r) \le s(j_{r+1})$ by IH.

Greedy can take $j_{r+1}$, but chooses smallest end time $f(i_{r+1}) \le f(j_{r+1})$

**Th** Greedy is optimal

Proof by contradiction

k - #jobs in i (Greedy), m - #jobs in j (optimal)

Suppose Greedy is not optimal $\Rightarrow k < m$.

By Lemma, $f(i_k) \le f(j_k)$

$\Rightarrow \exists j_{k+1}$ that starts after $j_k$ thus after $i_k$.

$\Rightarrow \exists i_{k+1}$ that is compatible with i. Contradiction!

2.1. Student Scheduling

Problem: Minimazing Lateness

Single resource processes one job at a time. Job j requires $t_j$ time and is due $d_j$.

If j starts at $s_j$, then it finishes at $f_j = s_j + t_j$. Lateness of j is $L_j = \max(0, f_j - d_j)$.

Minimize the maximum lateness $\max_j(L_j)$

Solution: Consider jobs in ascending order of deadline $d_j$

```
sort jobs by ascending deadline (d1 <= d2 <= … <= dn)
t <- 0
A <- {0}
for j <- 1 to n do
    t <- t + tj
    A <- A U {t}
return A
```

$O(n \log n)$

## 2.2. Exchange Argument

**Def.** Inversion in schedule – a pair of jobs s.t. $d_i < d_j$ but $j$ is scheduled before $i$

**Claim:** Swapping two adjacent inverted jobs reduce the number of invertions by 1 and does not increase the maximum lateness

$L$ – lateness before swap, $L'$ – after.

$L_k' = L_k \quad \forall k \neq i,j$

$L_i' \leq L_i$ ($i$ done earlier)

$L_j' = f_j' - d_j = f_i - d_j \leq f_i - d_i = L_i$

**Claim:** If a schedule (with no idle time) has an invertion, then it has one with a pair of inverted jobs scheduled consecutively (adjacent jobs).

Suppose there is an inversion.

There is a pair of jobs $i$ and $j$ s.t. $d_i < d_j$, but $j$ scheduled before $i$.

Walk through the schedule from $j$ to $i$.

Increasing deadlines = no inversions, at some point deadline decreases

**Th** Greedy schedule $S$ is optimal.

Suppose $S$ is not optimal.

Let $S'$ be an optimal schedule with the least inversions and no idle time $S \neq S'$

If $S'$ has no inversions, then the lateness is same as $S$. Contradiction!

If $S$ has an inversion, let $i$-$j$ be an adjacent inversion.

Swapping $i$ and $j$ does not increase the maximum lateness and decreases the number of invertions $\Rightarrow$ contradiction with definion of $S'$ $\Rightarrow$ $S$ is optimal

## 3.1. Connecting all machines

Problem: Creating the internet

$n$ machines and $m$ connections, each with cost $c(i)$

Find cheapest way to connect all the machines.

**Def** Spanning tree $T$ of graph $G = (V, E)$ – subset of edges $T \subseteq E$ s.t. graph $G' = (V, T)$ is strongly connected.

Minimum Spanning Tree (MST) of graph $G = (V, E)$ – spanning tree $T$ s.t. $c(T) = \sum_{e \in T} w(e)$ is minimal, $\nexists T'$ s.t. $c(T') < c(T)$

Solutions: Prim (Jarnik), Kruskal, Reverse Delete, Burovska

3.2. The Cut Property

S - subset of nodes V, e - min cost edge with exactly 1 endpoint in S => e ∈ MST.

3.3 Prim-Jarnik

Grow a cloud (S) by repeatedly adding the smallest edge out of the cloud.

```
function PrimJarnik(G)
    d[v] <- INF for all v in V
    s - arbitrary vertex in V
    d[s] <- 0; c <- 0
    while  not all vertices connected do
        m <- arg min d[v in V]
        c <- c + d[m]
        Remove m from V
        if d[m] = INF then
            return INF
        for e <- (m,u) in E for some u do
            if w(e) < d[u] then
                d[u] <- w(e)
    return c
```

$O(|E| \log|V|)$, even $\Theta(|E| + |V|\log|V|)$ with heap-based PQ

3.4. The Cycle Property

C - any cycle in G, e - max cost edge in C => e ∉ MST.

3.5. Kruskal

Sort edges by weight, repeatedly add smallest edge (from cut property), until there is a cycle, then discard it (from cycle property).

```
function Kruskal(G)
    sort E ascendingly (e1 <= e2 <= … <= em)
    T <- {}; i <- 0; k <- 0
    while k < |V| - 1 do
        if T U {ei} does not have a cycle then
            k <- k + 1
            T <- T U {ei}
    return T
```

$O(|E| \log|V|)$

3.6 Union-Find

Union-Find (Disjoint-Set) data structure - track elements partitioned into disjoint subsets; near-constant time merge (union) and contains (find)

```
MakeSet(n) - n initial sets, with 1 element each
    set <- array of size n, set[i] <- i
    rank <- array of size n, rank[i] <- 0
Find(x) - returns id of set containing x
    if set[x] != x then
        set[x] <- Find(set[x])
    return set[x]
Union(x,y) - merges set of x and y
    xSet <- Find(x)
    ySet <- Find(y)
    if xSet = ySet then
        return False
    else if rank[xSet] < rank[ySet] then
        set[xSet] <- ySet
    else
        set[ySet] <- xSet
        if rank[xSet] = rank[ySet] then
            rank[xSet] <- rank[xSet] + 1
    return True
```

4. K-clustering

Problem: Maximum Spacing K-Clustering

Divide objects into K non-empty groups.

$d(p_i, p_j) = 0$ iff $p_i = p_j$ otherwise $d(p_i, p_j) \geq 0$; $d(p_i, p_j) = d(p_j, p_i)$

Spacing - minimum distance between any pair of points in different clusters.

Solution: Single-list k-clustering

Form graph (without edges) on vertex set (n initial clusters).

Find closest pair of objects from different clusters and add edge

Repeat n-k times until exactly k clusters left.

Solution: Find MST, delete k-1 most expensive edges.

**Th** $C'$ - clustering $C_1', \dots, C_k'$ formed by deleting $k-1$ most expensive edges of a MST by Kruskal -> $C'$ - maximum spacing clustering

Proof

Spacing of $C'$ - length of $d'$ of the $(k-1)^{th}$ most expensive edge.

$C$ - arbitrary clustering $C_1, \dots, C_k$.

$p, q$ - in same cluster in $C'$ ($C_r'$), but different ones in $C$ ($C_s$ and $C_t$).

Some edge$(p', q')$ on $p$-$q$ path in $C_r'$ spans two different clusters in $C$.

All edges on this path have length $\le d'$ (Kruskal).

Spacing of $C \le d'$ since $p', q'$ in different clusters

$C$-arbitrary $\Rightarrow$ True for $\forall C \neq C'$

5. Huffman Encoding

Problem: Efficient Encoding

non-ambiguous binary encoding of text

**Def.** prefix code for set $S$ - $c: S \to \{0,1\}^+$ s.t. $\forall x,y \in S: x \neq y \to c(x)$ is not prefix of $c(y)$

Average Bits per Letter: $ABL(c) = \sum_{x \in S} f(x) \cdot |c(x)|$

Binary Tree:

- → Children uniquely identified by edge label (0 or 1).
- → Nodes labeled $x$ iff path from root labeled $c(x)$
- → Only leaves can have a label in a prefix code.

**Def.** full binary tree - node has either 0 or 2 children

**Lemma** $u, v$ - leaves in $T'$ and $depth_{T'}(u) < depth_{T'}(v)$ -> $f_u \ge f_v$

**Claim:** $\forall T, \exists T'$ optimal, where two lowest-frequency - sibling leaves at lowest level.

```
function Huffman(S)
    if ISI = 2 then
        return tree with root and 2 leaves
    else
        let y and a be the lowest frequency letters in S
        S' <- S - {y,z} U {w}, s.f. fw = fy + fz
        T' <- Huffman(S')
        T <- add two children y and z to leaf w in T'
```