CSC 373 - Algorithm Design, Analysis, and Complexity **Summer 2016**

Lalla Mouatadid

Greedy Algorithms: Interval Scheduling

Definitions and Notation:

A **graph** G is an ordered pair (V, E) where V denotes a set of vertices, sometimes called nodes, and E the corresponding set of edges (lines connecting the vertices). Formally $V = \{v_1, v_2, \ldots, v_n\}$ is the set of vertices and $E = \{(v_i, v_j) \in E$ means vertex v_i is connected to vertex $v_j\}$. The set of edges usually refers to a relationship between vertices. For instance, you can represent every person on Facebook as a vertex, and if two people are friends on Facebook, then their corresponding vertices are connected with an edge. We say two vertices v_i, v_j are **adjacent** if they are connected: $(v_i, v_j) \in E$. Given an edge $e = (v_i, v_j)$, we say that vertices v_i, v_j are **incident** to the edge e.

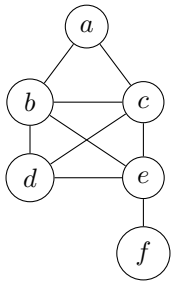
Given a graph $G(V, E)$, a **clique** $K(V', E')$ is a subgraph of G where $V' \subseteq V, E' \subseteq E$ and for all $u, v \in V', uv \in E'$. In other words *all* the vertices in K are pairwise adjacent. An **independent set** $H(\tilde{V}, \tilde{E})$ is also a subgraph of G where again $\tilde{V} \subseteq V, \tilde{E} \subseteq E$ and for all $u, v \in \tilde{V}, uv \notin \tilde{E}$. In other words *all* the vertices in H are pairwise non-adjacent, $\tilde{E} = \emptyset$.

A graph is **acyclic** if it does not contain a cycle. A **tree** is an acyclic connected graph. A collection of trees is called a **forest**

The **maximum independent set problem** asks for a largest **maximum** independent set in a graph $G(V, E)$.

Maximal vs. **Maximum**: Recall the difference between a **maximal** and a **maximum** solution S. We say that S is a **maximal** solution if there is no element $v \notin S$ that we can add to S and increase the size $|S|$ of S. On the other hand, S is a **maximum** solution if there exists no solution S' with $|S'| > |S|$. Clearly every maximum solution S is also maximal, but the reverse is not true.

Example: Consider the graph $G(V, E)$ below where $V = \{a, b, c, d, e, f\}, E = \{ab, ac, bc, bd, be, cd, ce, de, ef\}$. The set $\{b, c, d, e\}$ of vertices forms a clique. The set $\{a, e\}$ is an independent set. The set $\{a, d, f\}$ is also an independent set. Notice that $\{a, e\}$ is maximal whereas $\{a, d, f\}$ is a maximum independent set.



P is the class of problems that can be solved in polynomial time. We consider algorithms that have a polynomial runtime to be efficient, in general anything in the order of $n^{\mathcal{O}(1)}$, where n is the size of the input to the algorithm. On the other hand, problems that we do not know how to solve efficiently (we can only compute solutions in exponential time) fall in the class of **NP-hard** problems. We will come back to these classes towards the second half of the course.

Greedy Algorithms

Greedy Algorithms: At every iteration, you make a **myopic** decision. That is, you make the choice that is best at the time, without worrying about the future. And decisions are irrevocable; you do not change your mind once a decision is made.

With all these definitions in mind now, recall the music festival event scheduling problem. You have a pass to your favorite music fest, and want to attend/watch as many performances as possible. You like all the artists/bands equally; you just want to watch as many live shows as possible. But once you go to a performance, you cannot leave half way through to attend another one.
Clearly every performance has a start and a finish time, and you are given the schedule ahead of time. As we saw in class, we can think of each performance as a time interval (from its start time until it is over), and we can abstract this problem as the interval scheduling problem (ISP), defined below more formally:

Interval Scheduling Problem:
Input: A list of intervals $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$, where each interval I_i is defined by two integers $I_i = (s_i, f_i)$ with $s_i < f_i$. Two intervals I_i, I_j are **compatible**, i.e. **disjoint**, if they do not intersect ($f_i < s_j$ or $s_i < f_j$).
Output: A maximum subset of pairwise compatible (disjoint) intervals in $\mathcal{I}$.

A number of greedy heuristics we tried in class failed quickly and miserably. Heuristics such as the Greedy Early Start Time algorithm (sorting the intervals by nondecreasing start time $s_1 \leq s_2 \leq \ldots \leq s_n$), or the Greedy by Duration (sorting the intervals by nondecreasing duration $(f_1 - s_1) \leq (f_2 - s_2) \leq \ldots \leq (f_n - s_n)$) etc, but the **Early Finish Time** greedy algorithm (EFT) seemed to work, and we proved it is indeed optimal!

Algorithm 1 EFT

Input: A list of intervals $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$, where $I_i = (s_i, f_i)$ with $s_i < f_i$ for all $1 \leq i \leq n$
Output: A maximum subset of pairwise compatible intervals in $\mathcal{I}$

```
1: Sort the input list I of intervals in nondecreasing order of finish time.
2: f = 0
3: S_0 = ∅
4: for i ← 1 to n do
5:     if f < s_i then                          ▷ I_i is compatible with S_i
6:         S_i = S_{i−1} ∪ {I_i}
7:         f = f_i
8:     end if
9: end for
10: return S_n
```

Proof of optimality:

We will prove by induction that the solution returned by EFT is optimal. More precisely, we will show that at every step i, i.e. at every iteration, the partial solution S_i we created can be extended to some optimal solution $\mathcal{O}$. In other words, there exists an optimal solution $\mathcal{O}$ that contains S_i.

Proof. Every optimal solution contains the empty set and thus the claim holds for the base case $i = 0, S_0 = \emptyset$. Now suppose S_i can be extended to an optimal solution $\mathcal{O}$. It remains to show that S_{i+1} can also be extended to some optimal solution $\mathcal{O}^*$, not necessarily the same as $\mathcal{O}$. To prove this, we use an **exchange argument**.

There are two cases to consider. At step $i+1$, either no interval was added to S_i, in which case S_{i+1} can be

extended to $\mathcal{O}$, since $S_i = S_{i+1}$. Otherwise, let I be the interval added to S_i and let J be the first interval in $\mathcal{O}$ but not in S_i. If $I = J$ then we're done. Suppose then $I \neq J$. Since the algorithm selected I at step i, I must have the earliest finish time for all the remaining intervals in $\mathcal{I}$. In particular

$$f_I < f_J \text{ [1]} \tag{1}$$

Clearly I and J are compatible with every interval in S_i. Since $J \in \mathcal{O}$, J is compatible with every interval J' in $\mathcal{O}$ that comes after J. In particular

$$f_J < f_{J'} \tag{2}$$

Using (1) and (2), it follows that I is compatible with every interval J' in $\mathcal{O}$ that comes after J. So we can simply **exchange** J with I in $\mathcal{O}$, and obtain $\mathcal{O}^* = \mathcal{O} \cup \{I\} \backslash \{J\}$ where $\mathcal{O}^*$ is an optimal solution that extends S_{i+1}. □

Convince yourself of the following: Given a schedule $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$, construct a graph $G(V, E)$ as follows: For every interval I_i, create a vertex v_i, and two vertices are adjacent ($v_i v_j \in E$) if and only if their corresponding intervals overlap, then computing a maximum subset of pairwise compatible intervals is equivalent to computing a maximum independent set in G.

[1] we abuse notation here and use f_I, f_J instead of $f_k, f_{k'}$ (for $1 \leq k, k' \leq n$) to indicate the finish time of I, J respectively.