CS 218 Design and Analysis of Algorithms

Nutan Limaye

Indian Institute of Technology, Bombay
nutan@cse.iitb.ac.in

Module 1: Basics of algorithms

Typical running times: linear time

Given:	an array $A[1], A[2], \ldots, A[n]$
Output:	the value of the maximum element in the array

```
max ← $A[1]$
**for** $i = 2$ to $n$ **do**
    **if** $A[i] \geq$ max **then**
        max ← $A[i]$
    **end if**
**end for**
Output max
```

$O(n)$ comparisons.

Typical running times: $O(n \log n)$

Given:	an array $A[1], A[2], \ldots, A[n]$
Output:	output the array in a sorted order

Merge sort, heap sort. All perform $O(n \log n)$ comparisons.

Typical running times: quadratic, i.e. $O(n^2)$, time

Given:	n points, $p_1 = (x_1, y_1), p_2 = (x_2, y_2), \ldots, p_n = (x_n, y_n)$
Output:	i, j such that the distance between p_i, p_j is the minimum

```
min $\leftarrow (x_1 - x_2)^2 + (y_1 - y_2)^2$
**for** $i = 1$ to $n$ **do**
   **for** $j = i + 1$ to $n$ **do**
      $d \leftarrow (x_i - x_j)^2 + (y_i - y_j)^2$
      **if** min $> d$ **then**
         min $\leftarrow d$
      **end if**
   **end for**
**end for**
Output min
```

$O(n^2)$ comparisons.

Typical running times: polynomial, i.e. $O(n^k)$, time

Given: graph $G = (V, E)$

Output: $S \subseteq V$ s.t. no edge between any two nodes in S and $|S| = k$.

```
1: for each subset $S \subseteq V$ s.t. $|S| = k$ do
2:     flag ← true
3:     for each $u, v \in S$ do
4:         if $(u, v) \in E$ then
5:             flag ← false
6:         end if
7:     end for
8:     if flag = true then
9:         output $S$ and halt
10:    end if
11: end for
12: Output No Set Found
```

Running time

$\#$ subsets $= \binom{n}{k}$

for each subset S, the number of times lines $3 - 7$ run $= O(k^2)$.

$\therefore$ the total running time $\underline{\binom{n}{k} \times O(k^2)}$
$= ?$
$= O(n^k)$.

Polynomial if k is a constant.

For $k = 100$ it is very very large!

Greed is good and it works (sometimes...)

What is a greedy algorithm?

Builds a solution to the problem in small steps.

At each step it looks at the full information available so far.

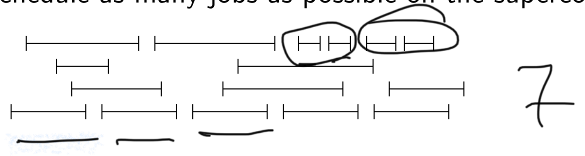
Takes the next step to optimise a local criterion.

Why should this optimise globally? Well, sometimes it can!

Interval Scheduling - Pick the one ending earliest

Problem Description

- A supercomputer on which jobs need to be scheduled.
- n jobs are specified using the start and finish times, i.e. $\forall i \in [n]$, $J_i = (s(i), f(i))$.
- We wish to schedule as many jobs as possible on the supercomputer.



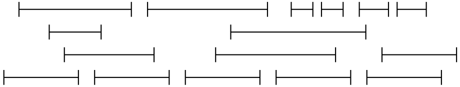
Problem Definition

Given a set $J = \{J_i = (s(i), f(i)) \mid i \in [n]\}$, find the largest subset $S \subseteq J$ such that no two intervals in S overlap.

Greedy approches

- Choose a job with the soonest starting time.
  Why should it work?

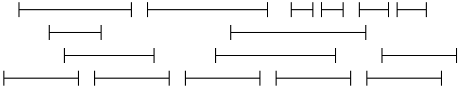
  Better to start using resources as soon as possible.



  This unfortunately does not work!

- Start with the smallest available job.
  Why should it work?

  Better to schedule as many jobs as possible.



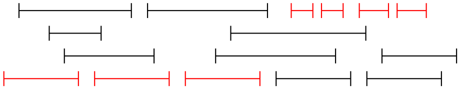
  Unfortunately this also does not work!

Greedy Approaches

- Choose a job with the earliest finish time.

  Why should it work?

  Better to keep as many resources free as possible.



  This seems to work!

- Why does it work?

  Let $\mathcal{A}$ denote the jobs selected by the algorithm.
  Let OPT denote the jobs selected by the optimal solution.

  We would like to say that the two are the same.
  It is enough to say that $|\mathcal{A}| = |\text{OPT}|$.

Correctness of the greedy heuristic

Let $i_1, i_2, \ldots, i_k$ be the jobs selected by $\mathcal{A}$.

Let $j_1, j_2, \ldots, j_m$ be the jobs selected by OPT.

Lemma

For all $r \leq k$, $f(i_r) \leq f(j_r)$.

For $r = 1$, by the choice of i_1, $f(i_1) \leq f(j_1)$.

Suppose $f(i_{r-1}) \leq f(j_{r-1})$.

When $\mathcal{A}$ picks its rth job, j_r is available for it to choose from.

Therefore, $f(i_r) \leq f(j_r)$.

If a job can be picked by OPT, it can also be picked by $\mathcal{A}$ at any given time.

This also shows that the algorithm $\mathcal{A}$ gives an optimal solution.

Running time of the algorithm

The running time of the algorithm can be analysed as follows.

Sorting the jobs by their finishing time take time $O(n \log n)$.

Scanning all the jobs through this sorted list takes time $O(n)$.

Therefore total running time $O(n \log n)$.

Minimum Spanning Subgraph

Problem description

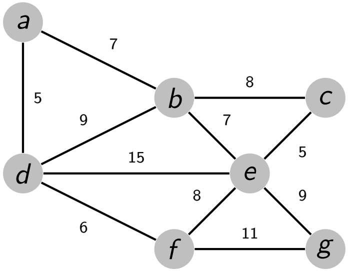
- Given an undirected connected graph $G = (V, E)$ and a cost function on the edges $c : E \to \mathbb{Z}^+$.
- Find a subset $T \subseteq E$ such that

  T must span all the vertices,

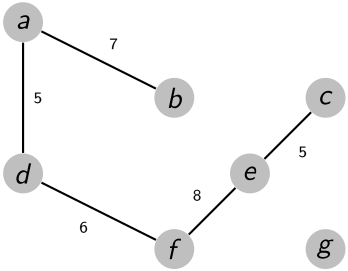
  T must be connected,

  T must be the least cost such set.

Graph G with edge costs



non-example for T



Minimum Spanning Subgraph

Problem description

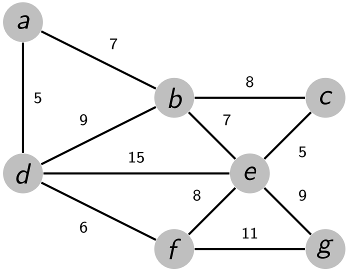
- Given an undirected connected graph $G = (V, E)$ and a cost function on the edges $c : E \to \mathbb{Z}^+$.
- Find a subset $T \subseteq E$ such that

  T must span all the vertices,

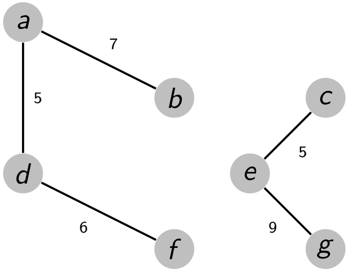
  T must be connected,

  T must be the least cost such set.

Graph G with edge costs



non-example for T



Minimum Spanning Subgraph

Problem description

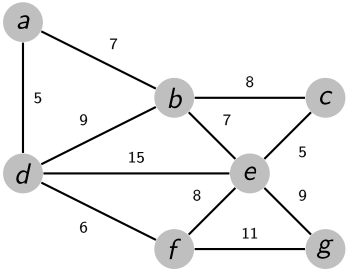
- Given an undirected connected graph $G = (V, E)$ and a cost function on the edges $c : E \to \mathbb{Z}^+$.
- Find a subset $T \subseteq E$ such that

  T must span all the vertices,

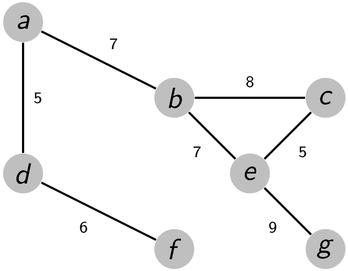
  T must be connected,

  T must be the least cost such set.

Graph G with edge costs



non-example for T



a
b
c
d
e
f
g
7
5
8
7
5
6
9