Algorithm Theory - Exercise Class

Exercise Lesson 3

Albert-Ludwigs-Universität Freiburg

Philipp Schneider

Algorithms and Complexity - Professor Dr. Fabian Kuhn

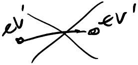
Organizational Matters

- English Tutorial in Room 101-02-016/18.
- Next exercise by Mohamad Ahmadi in English and in this room.
- **Email subject:** `AlgoTheo1718_[Sheet-Number]`

Pseudocode: Standard Operating Procedure

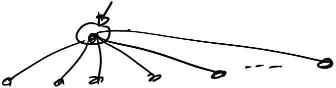
- Do not 'program'
- *Most* of the time you don't need...
  - ... classes
  - ... subprocedures for easy tasks (describe them instead)
  - ... subprocedures replicating mathematical operations e.g. $\bigcup_{x \in Set} \{f(x)\}$
  - ... brackets around forks, loops
- Algorithm Theory $\neq$ Software Engineering
- Concentrate on your strategy
- Emphasis on the analysis of correctness and runtime
- 'Neglect' implementation details, wherever possible
- It's perfectly fine to describe an algorithm with text

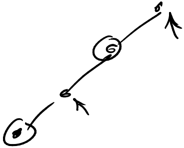
MIS vs maxIS



Let $G = (V, E)$ be a graph.

- $V' \subset V$ *independent* if for any nodes $u, v \in V'$ it holds that $\{u, v\} \notin E$.
- V' *maximal* independent (MIS) if V' is independent and there can no node be added without violating independence.
- V' *maximum* independent (maxIS) if V' is independent and $|V'|$ is maximum among all independent sets of G.





Greedy - MaxIS

Devise an *efficient* algorithm that *computes a maximum independent set* in a rooted tree (for node v let $C(v) :=$ children of v in T).

$= (V,E)$

$n := |T|$

Algorithmus 1 : `GreedyTreeMaxIS`(v,T)

---

for $u \in C(v)$ **do**
  `GreedyTreeMaxIS`(u,T)

if $\forall u \in C(v) : u \notin S$ **then**
  add v to S `// S global`

---

leaf is added to S since $\underset{\emptyset}{\forall}(\cdot) :=$ true

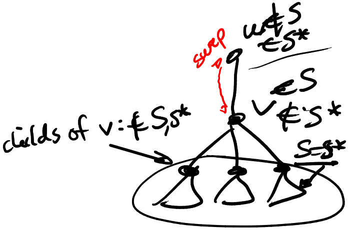
Runtime: we have $|V|$ calls of GreedyTreeMaxIS.
that is: $O(n)$ many calls.
Each node occurs once in the if-clause where we test whether the node is in S.
Amortized: $O(n)$ for the tests $u \notin S$.

Correctness: Independence ✓ (inductively). Show that S is maximum.
Let S^* a max IS. Let $v \in V$ where S and S^* differ. Let v be such that S and S^* do not differ in the subtrees attached to v. Greedy: $v \in S, v \notin S^*$.
parent w of v: $w \in S^*, w \notin S$. $S^{(1)} := (S^* \cup \{v\}) \setminus \{w\}$, $|S^*| = |S^{(1)}|$

repeat this process: $|S^*| = |S^{(1)}| = |S^{(2)}| = \dots = |S|$

independent

$\Rightarrow |S| = |S^*| \Rightarrow |S|$ is maximum.

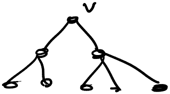


Dynamic Programming - MaxIS

Devise an *efficient* algorithm that that uses dynamic programming and *computes the size of a maximum independent set* in a rooted tree.

$$\underline{s(v)} = \max\left\{ \underline{\sum_{u\in C(v)} s(u)},\ \underline{1+\sum_{u\in C(v)}\sum_{w\in C(u)} s(w)} \right\}$$

size of max IS of subtree with root v.



Algorithmus 2 : `TreeMaxIS(v, T)`

if $memo[v] \neq \perp$ **then**
  return `memo`$[v]$

$c \leftarrow \sum_{u\in C(v)}$ `TreeMaxIS(u, T)`
$g \leftarrow 1+\sum_{u\in C(v)}\sum_{w\in C(u)}$ `TreeMaxIS(w, T)`

`memo`$[v] \leftarrow \max\{c,g\}$

return `memo`$[v]$

Base cases: $\sum_{\emptyset}(\cdot) := 0$ ○ $g=1$, $c=0$

Runtime:

Max. $O(n)$ calls of TreeMaxIS due to memorization.
Globally seen, we have $O(n)$ summations, since each node occurs at most twice in a sum once due to the parent once due to the grand-parent.

Exercise 1: Dynamic Programming - MaxIS

(a) Devise an *efficient* algorithm that uses the principle of dynamic programming and *finds a maximum independent set* in a rooted tree.

$C(v) := \{u \mid u \text{ child of } v\}$

Algorithmus 3 : TreeMaxIS(v)

if $memo[v] \neq \perp$ **then**
  return memo[v]

$c \leftarrow \sum_{u \in C(v)}$ TreeMaxIS(u)
$g \leftarrow \underline{1} + \sum_{u \in C(v)} \sum_{w \in C(u)}$ TreeMaxIS(w)

if $g > c$ **then**
  add v to $\boxed{\text{maxIS}}$

$O(1)$

memo[v] $\leftarrow \max\{c, g\}$

return memo[v]

Runtime: $O(n)$ ✓

Correctness: maxIS is maximum ✓

claim: For subtree $T' = (V', E')$ of T we have that maxIS $\cap V'$ is maximum independent in T'.

Induction Bases: $g = 1$, $c = 0$ ✓.

maxIS $\ni v$

u

Induction step: $v \notin$ maxIS. independence is fullfilled. Consider $v \in$ maxIS. Assume child $u \in C(v)$ is in maxIS. We know $\underline{g > c}$ ($v \in$ maxIS).

Since $u \in$ maxIS $s(u) > \sum_{u \in C(u)} s(w)$

$\underbrace{\sum_{u \in C(v)} s(u)}_{c} > \underbrace{\sum_{u \in C(v)} \sum_{w \in C(u)} s(u)}_{g-1}$

$c > g - 1$
$\Rightarrow c \geq g$ ↯ $\Leftarrow$

Exercise 1: Dynamic Programming - MaxIS

(b) Prove that your algorithm is correct, i.e. returns a maximum independent set and prove that it has the claimed runtime.

UNI
FREIBURG

Worst Case Analysis - Fibonacci Heaps

Show that in the worst case (a) the `delete-min` operation can require time $\Omega(n)$ *for an arbitrary n.*

insert n keys

Rootlist:

delete_Min: $\rightsquigarrow$ consolidate

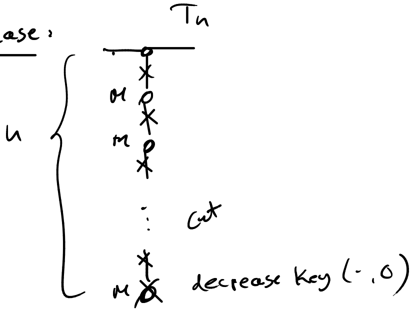
$\rightsquigarrow$ we touch every node once

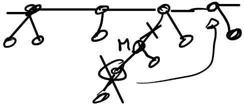
$\rightsquigarrow$ $\Omega(n)$

Worst Case Analysis - Fibonacci Heaps

Show that in the worst case (b) the `decrease-key` operation can require time $\Omega(n)$ *for an arbitrary n.*

worst case:



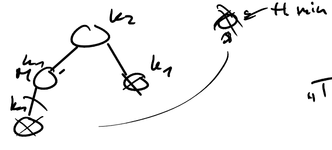


$\Rightarrow \Omega(n)$

T_n is a valid Fibonacci-Heap: Inductively

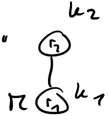
Base: we want T_2: 

insert keys $k_2, k_?, k_?, k_1, 0$ with $0 < k_2 < k_1$

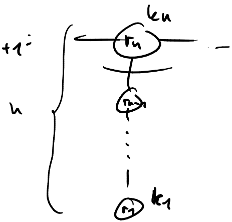
delete-Min: 

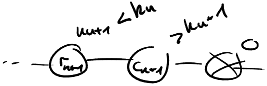
$\emptyset$ = H.min

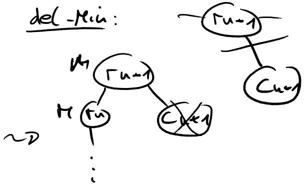
dec.Key(, 0) del-Min. $\rightsquigarrow$



step: $T_n \rightsquigarrow T_{n+1}$:



insert: 

del-Min: 

Amortized Analysis - Counting

We execute n increment operations on a binary number starting from 0. Flipping the i^{th} bit b_i now has a cost 2^i.

(a) Show that the amortized cost is super-constant (i.e. in $\omega(1)$).

Amortized Analysis - Counting

We execute n increment operations on a binary number starting from 0. Flipping the i^{th} bit b_i now has a cost 2^i.

(b) Show that the amortized cost is $\mathcal{O}(\log n)$.

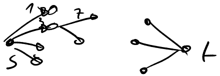
Graph Algorithms: Max-Flow

Consider the Max-Flow Problem of a directed graph $G = (V, E)$ with capacities $c : E \to \mathbb{N}_0$.
Instead of just one source and one sink, you are given k sources $s_1, \ldots, s_k$ and ℓ sinks $t_1, \ldots t_\ell$. The flow function $f : E \to \mathbb{N}_0$ must satisfy the properties

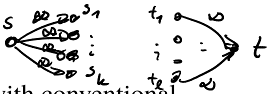
Capacity constraints: $f(e) \leq c(e)$, for all $e \in E$

Flow Conservation: $\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$, for all $v \in V \setminus \{s_1, \ldots, s_k, t_1, \ldots, t_\ell\}$

The value of a flow is



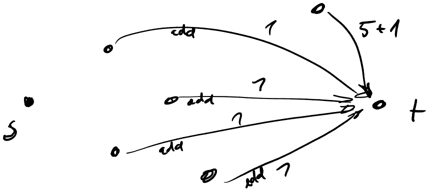
$$|f| = \sum_{i=1}^{k} \sum_{e \text{ out of } s_i} f(e) = \sum_{j=1}^{\ell} \sum_{e \text{ into } t_j} f(e)$$



Show that you can reduce this problem in order to solve it with conventional means (e.g. the algorithm of Ford-Fulkerson from the lecture).

Graph Algorithms: Max-Leaky-Flow

Consider the Max-Flow problem with one source s and one sink t, but each node leaks an amount of 1 unit. This means that if c is flowing into v then only $\max(0, c-1)$ is flowing out. The leaking flow is counted towards the value of the flow, though. Can you reduce this problem as well?



source s
sink t

For $G=(V,E)$ cap. $c: E \to \mathbb{R}^+$

$G'=(V,E')$ cap. $c': E' \to \mathbb{R}^+$

$E' = E \cup \{\{u,t\} \mid u \in V \setminus \{t\}\}$

$c'(e) := 1$ for $e \in E' \setminus E$

$c'((u,t)) := c((u,t)) + 1$ for $(u,t) \in E$

$c'(e) := c(e)$ else