Rheinische Friedrich-Wilhelms-Universität Bonn
Mathematisch-Naturwissenschaftliche Fakultät

Theoretical Aspects of Intruder Search

MA-INF 1318 Manuscript Wintersemsester 2015/2016

Elmar Langetepe

Bonn, 19. October 2015

The manuscript will be successively extended during the lecture in the Wintersemester. Hints and comments for improvements can be given to Elmar Langetepe by E-Mail `elmar.langetepe@informatik.uni-bonn.de`. Thanks in advance!

Lemma 21 *Any contiguous monotone strategy for T' can be translated to a contiguous monotone strategy for T with the same number k of agents.*

Proof. Let $e' = (x, y)$ and $e'' = (y, z)$ be links stemming from the extension of a link e. If q guards move from x to y or z to y, they stay in there place in T. If q guards move from y to x or from y to z, they will move from z to x or from x to z in T, respectively. □

The other way round, any strategy for T is also a strategy for T'.

Lemma 22 *Any contiguous monotone strategy for T with k agents can be translated to a contiguous monotone strategy for T' with the same number k of agents.*

Proof. A move along an edge e in T is splitted into two moves along e' and e'' in T'. If the move clears e, then $q \geq w(e)$ have traversed e. From the construction q searchers are also enough for $w(e) = w(e') = w(e'')$ and the weight $w(e)$ of the intermediate vertex. □

We collect our results:

Proof of Theorem 17: From Lemma 21 we conclude $\mathrm{cs}(T') \leq \mathrm{cs}(T)$. From Lemma 18 we obtain a connected crusade of frontier $\leq \mathrm{cs}(T)$ in T'. From Lemma 19 we conclude that there is a progressive connected crusade of frontier $\leq \mathrm{cs}(T)$ in T'. From Lemma 20 we obtain a monotone contiguous search strategy using $\leq \mathrm{cs}(T)$ guards in T' and we can assume that all searchers are initially at a single starting vertex v_1. From Lemma 22 we conclude that there is also an optimal monotone contiguous search strategy that starts with all guards in a single vertex.

2.2.5 Designing a monotone strategy for unit weights

By Theorem 17 we can start strategy from a single vertex v and we can consider monotone strategies. Therefore, we design an optimal strategy for any starting vertex v and for the rooted tree T_v we compute the minimum number, $\mathrm{cs}(T_v)$, of agents required for starting in v. Finally we have $\mathrm{cs}(T) = \min_{v \in T} \mathrm{cs}(T_v)$.

An optimal monotone strategy for computing, $\mathrm{cs}(T_v)$, will also give an ordering all vertices z of T_v, stating which subtree, say $T_v(z)$, of T_v w.r.t. root v is fully cleared first. For this we can also consider the subtree $T_v(z)$ alone with root z and ask for $\mathrm{cs}(T_v(z))$ for short and an optimal monotone strategy.

We denote the children of the vertex z of the subtree $T_v(z)$ of T_v by $z_1, \ldots, z_d$ w.r.t. the order $\mathrm{cs}(T_v(z_i)) \geq \mathrm{cs}(T_v(z_{i+1}))$ for $i = 1, \ldots, d-1$. An example is given in Figure 2.11. Now, we can prove the main structural result. Unfortunately, there is a flaw in the proof of Barrière at al. and we can only proof the statement for unit weighted trees. The flaw is precisely marked in the proof below.

Lemma 23 *Let $z_1, \ldots, z_d$ be the $d \geq 2$ children of a vertex z in T_v and assume that $cs(T_v(z_i)) \geq cs(T_v(z_{i+1}))$ for $i = 1, \ldots, d-1$. We have*

$$cs(T_v(z)) = \max\{cs(T_v(z_1)), cs(T_v(z_2)) + w(z)\} \tag{2.5}$$

it the tree T is a tree with unit weights.

Proof. We can assume that $\mathrm{cs}(T_v(z)) \geq \mathrm{cs}(T_v(z_1))$ holds because we have to clear $T_v(z_1)$ before clearing $T_v(z)$. If in Equation 2.5 $\mathrm{cs}(T_v(z_1) \geq \mathrm{cs}(T_v(z_2) + w(z)$ holds, we can clear $T_v(z)$ by setting $w(z)$ on z and clear all $T_v(z_i)$ by $\mathrm{cs}(T_v(z_1)$ agents but $T_v(z_1)$ last. Note that also $w((z, z_i)) \leq w(z_i) \leq \mathrm{cs}(T_v(z_i)$ for all i for moving back from subtrees to z. Altogether, $\mathrm{cs}(T_v(z_1)$ agents are required and they are sufficient.

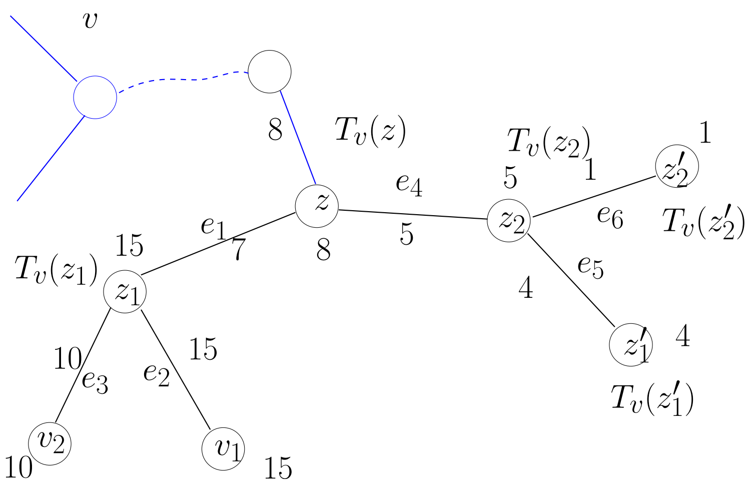


Figure 2.11: The rooted tree T_v has two subtrees $T_v(z_1)$ and $T_v(z_2)$ at vertex z. We have $\mathrm{cs}(T_v(z_1)) = 25$ and $\mathrm{cs}(T_v(z_2) = 6$ and $\mathrm{cs}(T_v(z_2)) + w(z) = 14 < 25 = \mathrm{cs}(T_v(z_1))$ which means that $\mathrm{cs}(T_v(z)) = 25$ holds. We leave $w(z)$ agents at z and clean $T_v(z_2)$ first. In $T_v(z_2)$ the same situation occurs, here $\mathrm{cs}(T_v(z'_1)) = 4$ and $\mathrm{cs}(T_v(z_2)) = 1$ but $\mathrm{cs}(T_v(z'_2)) + w(z_2) = 6 > 4 = \mathrm{cs}(T_v(z'_1)$. Therefore we require $\mathrm{cs}(T_v(z'_2)) + w(z_2) = 6$ agents, first we clean $T_v(z'_2)$ by 1 agent and block z_2 by 5 agents. Then we clean $T_v(z'_1)$ by 6 agents.

So let us assume that in Equation 2.5 $\mathrm{cs}(T_v(z_1)) < \mathrm{cs}(T_v(z_2)) + w(z)$ holds. We would like to prove that $\mathrm{cs}(T_v(z_2)) + w(z) - 1$ agents are not sufficient. We consider two cases:

1. **$T_v(z_2)$ is cleared before $T_v(z_1)$:** While $\mathrm{cs}(T_v(z_2))$ agents clear $T_v(z_2)$ there are only $w(z) - 1 = 0$ agents left for blocking a vertex in $T_v(z_1)$. Recontamination!

2. **$T_v(z_1)$ is cleared before $T_v(z_2)$):** While $\mathrm{cs}(T_v(z_1))$ agents clear $T_v(z_1)$ there are no more than $w(z) - 1 = 0$ agents left for blocking a vertex in $T_v(z_2)$ (because $\mathrm{cs}(T_v(z_1)) = \mathrm{cs}(T_v(z_2)))$. Recontamination!

The above statement do not hold for general weighted trees, because the fact that one only partially decontaminates $T_v(z_2)$ or $T_v(z_1)$ is not taken into account. For example, consider the vertex, say v with weight 5 in the center of Figure 2.12. and let z_1, z_2, and z_3 be the children of v below v from right to left. We have $\max\{\mathrm{cs}(T_x(z_1)), \mathrm{cs}(T_x(z_2)) + w(z)\} = \max\{8, 7 + 5\} = 12$ but we can recontaminate the subtree by 10 agents only, if we first clean z_3, leaving 5 agent at v. Then only clean vertex z_2 with one agent and leave this agent there. Then we clean $T_x(z_1)$ with the remaining 9 agents, and finally return to z_3 for the last part.

So $\mathrm{cs}(T_v(z_2)) + w(z)$ are required and are also sufficient by occupying z with $w(z)$ and clearing all $T_v(z_i)$ by $\mathrm{cs}(T_v(z_2))$ agents but $T_v(z_1)$ last with $\mathrm{cs}(T_v(z_2)) + w(z)$ agents. Note that also $w((z, z_i)) \leq w(z_i) \leq \mathrm{cs}(T_v(z_i))$ for all i for moving back from subtrees to z. □

The consequence of the above Lemma is, that we can compute $\mathrm{cs}(T_v)$ in $O(n)$ time by recursively applying Equation 2.5. Alternatively, we can start from the vertices.

Exercise 13 *Compute $cs(T_{v_4})$ for the tree in Figure 2.5 by the above recursive process.*

Corollary 24 *For a unit weighted tree T of size n and for a given starting vertex v we can compute the optimal monotone contiguous strategy starting at v in $O(n)$ time. An overall optimal contiguous strategy can be computed in $O(n^2)$.*

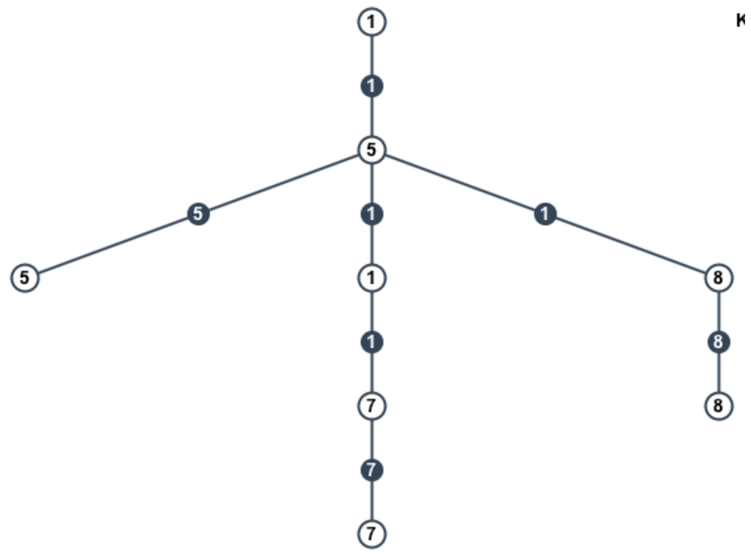


Figure 2.12: The flaw in the prove of Barriére et al. The recursion $\text{cs}(T_v(z)) = \max\{\text{cs}(T_v(z_1)), \text{cs}(T_v(z_2)) + w(z)\}$ does not hold for arbitrary weighted trees.

2.2.6 Computing an optimal contiguous Intruder Search Strategy for unit weights

We consider a message based algorithm that compute the optimal number of agents required for any starting vertex v.

The following local recursive labeling $\lambda_x(e)$ for the links $e = (x, y)$ adjacent to x will be sufficient. Let $e = (x, y)$ be a link incident to x.

1. If y is a leaf, set $\lambda_x(e) = w(y)$.
2. Otherwise, let d be the degree of y and let $x_1, \ldots, x_{d-1}$ be the incident vertices of y different form x. Let $\lambda_y(y, x_i) =: l_i$ and $l_i \geq l_{i+1}$. Then,

$$\lambda_x(e) := \max\{l_1, l_2 + w(y)\}\,.$$

For any link $e = (x, y)$ we will have two labels $\lambda_x(e)$ and $\lambda_y(e)$. By a *messages sending* technique, we can compute the labels $\lambda_x(e)$ and $\lambda_y(e)$ for alle edges $e = (x, y)$ in overall linear time. Note that we interpret any link $e = (x, y)$ as undirected, which means that $(x, y) = (y, x) = e$, more formally we could have used a notion $e = \{x, y\}$.

The message sending algorithm works as follows:

1. Start with the leaves and for any leaf y and for $e = (x, y)$ send a message $l = w(y)$ to x. After receiving this messages, x sets $\lambda_x(e) = l$.
2. Consider a vertex y of degree d that has received at least $d-1$ messages l_i from the incident certices $x_1, \ldots, x_{d-1}$ and let x be the remaining incident vertex. Let $l_i \geq l_{i+1}$. Send a message $l = \max\{l_1, l_2 + w(y)\}$ to x, after receiving the message x, set $\lambda_x((x, y)) = l$.

The above process can be applied sequentially, starting from the leaves. The process will not stop until we have send a message from x to y and y to x along any edge $e = (x, y)$. The process ends and in total $O(n)$ messages have been transmitted. An example is given in Figure 2.13. Keep in mind that we set $\lambda_x(e)$ meaning that x has received a message from e.

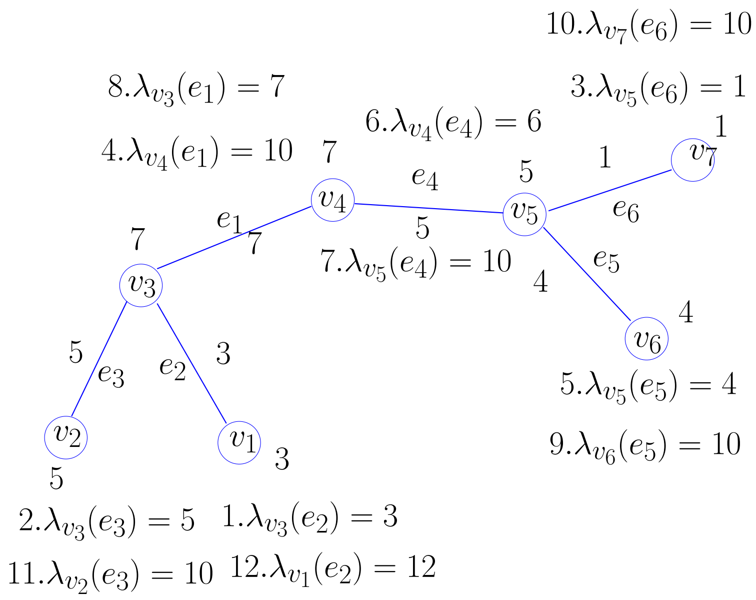


Figure 2.13: The message sending algorithm can easily work sequentially.

Lemma 25 *The links of a tree T can be labeled with labels λ_x by the above message sending algorithm by $O(n)$ messages in total.*

Finally, we would like to prove that for an edge $e = (x, y)$ the labeling algorithm indeed computes $\text{cs}(T_x(y))$ for the rooted tree T_x and its direct neighbor y. Note, that we can only proof the result for unit weighted trees.

Lemma 26 *For a unit weighted tree $T = (V, E)$ and an edge $e = (x, y) \in E$ we have $cs(T_x(y)) = \lambda_x(e)$.*

Proof. The proof goes by induction on the height $h(y)$ of $T_x(y)$. If y is a leaf we have $\lambda_x(e) = w(y)$ for $h(y) = 0$. The statement holds.

Assume that the statement holds for $0 \leq h(y) < k$ and consider $h(y) = k$. For edge $e = (x, y)$ let $x_1, \ldots, x_d$ be the $d \geq 1$ be the children of y in $T_x(y)$ and assume that $\lambda_y((y, x_i)) \geq \lambda_y((y, x_{i+1}))$ holds for $i = 1, \ldots, d-1$. We also have $T_y(x_i) = \lambda_y((y, x_i)$ by induction hypothesis and $T_y(x_i) = T_x(x_i)$ by definition. Therefore we also have $\text{cs}(T_x(x_i)) \geq \text{cs}(T_x(x_{i+1}))$ for $i = 1, \ldots, d-1$.

In Lemma 23 the recursion Equation 2.5 for $T_x(y)$ is exactly the same as step 2. $\lambda_x((x, y))$ for in the labeling process 2.2.6.

Therefore, we conclude $\text{cs}(T_x(y)) = \lambda_x(y)$. □

Finally, we have to compute the optimal number of agents and also a corresponding strategy. The first part can done as follows. We compute the minimum number of agents, $\mu(v)$ required for starting at a vertex v in the tree T.

For this we order all $\lambda_v((v, x_i)$ for all $i = 1, \ldots, d$ incident edges (v, x_i) so that $\lambda_v((v, x_i)) \geq \lambda_v((v, x_{i+1}))$ and compute

$$\mu(v) = \max\{\lambda_v((v, x_1)), \lambda_v((v, x_2)) + w(v)\} \,. \tag{2.6}$$

See for example the computation of $\mu(v_3)$ and $\mu(v_5)$ in Figure 2.14.

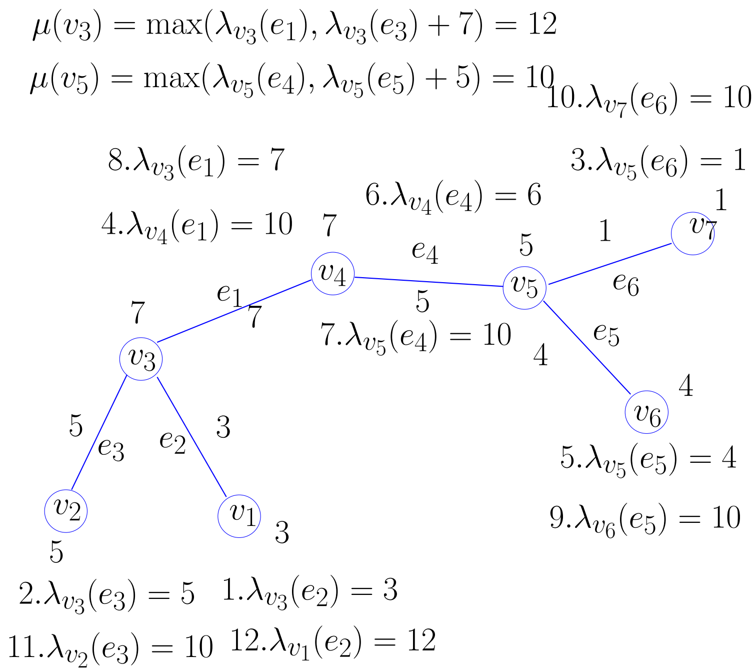


Figure 2.14: Computing $\mu(v) = \max\{\lambda_v((v, x_1)), \lambda_v((v, x_2)) + w(v)\}$ and the minimal $\min_{v \in V} \mu(v) = \mathrm{cs}(T)$ gives an optimal strategy at least for unit weighted trees.

Altogether, we have $\mu(v) = \mathrm{cs}(T_v)$ and $\min_{v \in V} \mu(v) = \mathrm{cs}(T)$. For the movements of the agents we choose the vertex v that attains a minimum $\mu(v)$ and apply a strategy as induced by the values λ_y. We traverse T_v in increasing order of the values λ_y.

For example, in Figure 2.14 $\mu(v_5) = 10$ gives the minimal number of agents required and we start with 10 agents in v_4 w.r.t. decreasing numbers λ_{v_5}. Thus, first 1 agenst move along e_6 and back to v_5, then 4 agents move along e_5 and back to v_5. After that 10 agents move along e_4 and so on.

Theorem 27 *On optimal contiguous strategy for a unit weighted tree $T = (V, E)$ can be computed in $O(n)$ time and space.*

Proof. The number of message required is given by the above considerations. For calculating the messages (and also the values $\mu(x)$) afterwards, we only have to register the greatest three entries $\lambda_v(e)$ for any v. This can be done successively. For any new message we can adjust the greatest three entries in constant time. □

2.2.7 Lower and upper bound for the contiguous search

For a given tree, T_n with n nodes we are asking for the $\max_n \mathrm{cs}(T_n) =: \mathrm{cs}(n)$. For convenience we consider the unit weighted case, where all weights are equal to 1. We will prove the following Theorem.

Theorem 28 *For unit weights and for any number of vertices n, we have $\lfloor \log_2 n \rfloor - 1 \leq cs(n) \leq \lfloor \log_2 n \rfloor$.*

We prove each inequality of the Theorem separately by the following lemmata:

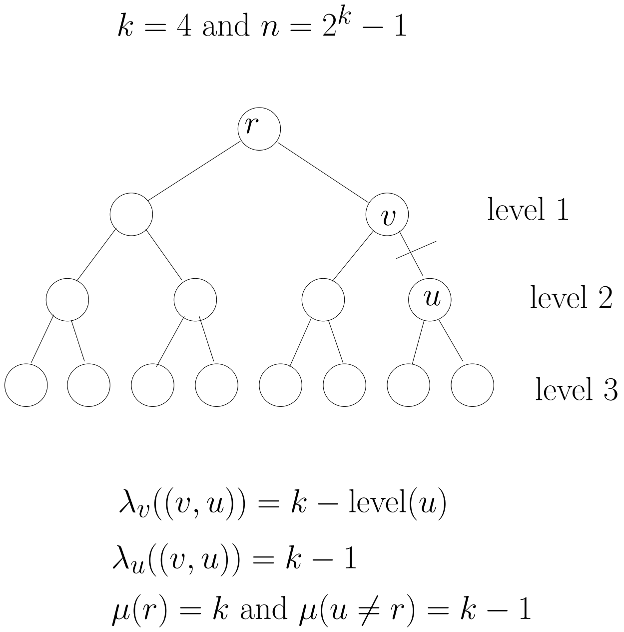


Figure 2.15: For $k = 4$ and $n = 2^k - 1$ and T_n as the full binary tree, we conclude $\text{cs}(T_n) = k - 1$ which gives the bound.

Lemma 29 *For every $n \geq 1$ we find trees T_n with $cs(T_n) \geq \lfloor \log_2(\frac{2}{3}(n+1)) \rfloor \geq \lfloor \log_2 n \rfloor - 1$.*

Proof. We consider a rooted tree T with root r and for any vertex u let the *level of* u denote the distance from r to u. If n equals $2^k - 1$ we choose a complete binary tree and show that $\text{cs}(T_n) = k - 1 = \log_2(n+1) - 1 \geq \log_2 \lfloor (\frac{2}{3}(n+1)) \rfloor$ agents are required by considering the values $\lambda_v(e)$. See also Figue 2.15.

- We have $\lambda_v((v,u)) = k - i$ and $\lambda_u((v,u) = k - 1$, for any vertex u of level $i > 0$ and its parent node v w.r.t. r. This can be easily seen by induction. The last value stem from the fact that we have to clean a complete tree with $2^{k-1} - 1$ vertices by starting from the root node.
- We have $\mu(u) = k - 1$ for any $u \neq r$ and $\mu(r) = k$, which gives the bound.

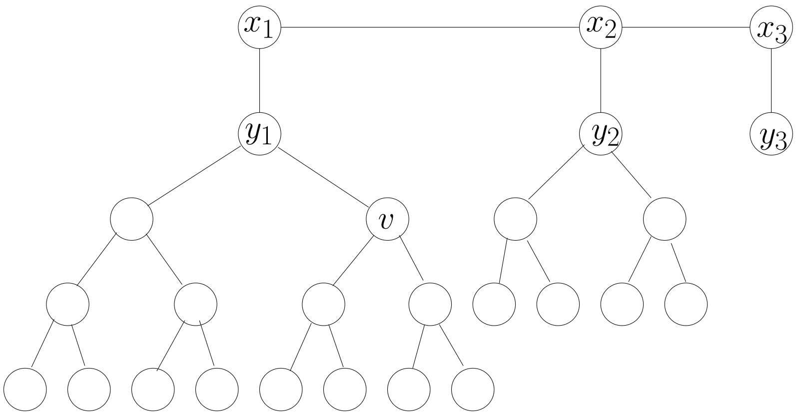
Now, for $n \neq 2^k - 1$ consider the binary representation $n = \sum_{i=1}^{r} 2^{\alpha_i}$ with $\alpha_1 > \alpha_2 > \cdots > \alpha_r$. For example consider $n = 11010$ in binary representation with $\alpha_1 = 4$,$\alpha_2 = 3$, $\alpha_3 = 2$. We build a chain with vertices $x_1, x_2, \ldots, x_r$ and for any x_i we build an edge to a complete binary tree T_{α_i} of size $2^{\alpha_i} - 1$ as depicted in Figue 2.16.

This means that we have n vertices in total. We conclude that α_1 agents are required. This holds if we start somewhere outside T_{α_1} because we visit the root of T_{α_1} at some point. If we start inside T_{α_1} (for example in a leaf) we require $\alpha_1 - 1$ agents for T_{α_1} at most but at the root node y_i of T_{α_1} we can assume that we have to place an additional agent that blocks the recontamination from x_1.

For this we assume that we require at least $\alpha_1 - 1 = \lambda_{y_1}((y_1, x_1))$. In our example this is the case because cleaning T_{α_2} from the root requires $\alpha_2 = \alpha_1 - 1$ agent. (If this is not the case $\alpha_1 - 1$ agents are enough in total, but also n is small enough in this case so that we can also conclude $\alpha_1 - 1 \geq \lfloor \log_2(\frac{2}{3}(n+1)) \rfloor$ which is an Exercise for the cases $\alpha_2 \leq \alpha_1 - 2$).

Altogether in the above case, we have $2^{\alpha_1} - 1 < n < 2^{\alpha_1+1} - 1$ and require $\text{cs}(T_n) = \alpha_1 \geq \log_2(n+1) - 1 \geq \log_2 \lfloor (\frac{2}{3}(n+1)) \rfloor$ agents in total which gives the conclusion. $\square$

$$n = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 11010$$



$$\lambda_{y_1}((v, y_1)) = \alpha_1 - 1$$
$$\lambda_{y_1}((x_1, y_1)) = \alpha_2 + 1 = \alpha_1$$

Figure 2.16: A tree T_n with $n = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 26$ vertices, requires α_1 agents.

Exercise 14 *Discuss the remaining case in the above proof. That is $\alpha_2 < \alpha_1 + 1$. Consider $\alpha_2 = \alpha_1 + 2$ and $\alpha_2 < \alpha_1 + 2$ separately.*

On the other hand we show that $\lfloor \log_2 n \rfloor$ agents are always sufficient.

Lemma 30 *For every $n \geq 1$ and unit weights, $\lfloor \log_2 n \rfloor$ agents are sufficient for a contiguous search strategy.*

Proof. We consider a tree T_r with n vertices and $\mu(r) = \text{cs}(T)$. Now we simplify this so that it becomes a complete binary tree T'_r w.r.t. r with $\text{cs}(T_r) = \text{cs}(T'_r)$ by the following rules, which will be applied until none of them is applicable any more. The children/parent relation in the tree is considered w.r.t. r.

1. For a node x and its $d > 2$ children $x_1, x_2, \ldots, x_d$ ordered by $\text{cs}(T_r(x_i)) \geq \text{cs}(T_r(x_{i+1}))$ remove all $T_r(x_i)$ for $i > 2$.

2. For a node x with two children x_1 and x_2 and $\text{cs}(T_r(x_1)) > \text{cs}(T_r(x_2))$, remove $T_r(x_2)$.

3. For a node $x \neq r$ with only one child x_1, remove x and connect x_1 to the parent of x.

4. If there are more than two vertices left, and r has only one child x_1, remove x_1 and connect the children of x_1 to r.

First, the number of agents required for T'_r and T_r are the same, because the computation of $\mu(r)$ in T_r makes use of eaxctly the same values. Note that the weights of the vertices are restricted to one, therefore rule 2. is also correct by $\text{cs}(T_r(x_1)) \geq \text{cs}(T_r(x_2)) + 1$. Cancelling a vertex with one child has no influence.

Second, we show that T'_r is a complete binary tree rooted in r. The first rule and the second rule returns a tree that has internal nodes with at most 2 children. Rule three deletes internal nodes with one child except for the root. Rule 4 make the root have 2 or 0 children.

Thus, we have a binary tree whose internal nodes have degree excactly 2. Finally, we show that the tree is complete. Let x be a node such that the subtree T'_x at x is not complete and there is no other subtree in T'_x with this property. This means that the children x_1 and x_2 of x in T'_r define complete subtree T'_{x_1} and T'_{x_2} of different size. Thus, rule 2 can be applied which gives a contradiction. □