UNIVERSITÄT BASEL

Departement Mathematik und Informatik
Philosophisch-Naturwissenschaftliche Fakultät
Universität Basel

Solving the Traveling Tournament Problem with Heuristic Search

Präsentation Bachelorarbeit

12. Februar, 2015

Patrik Dürrenberger
patrik.duerrenberger@stud.unibas.ch

Einleitung

- Ziel der Bachelorarbeit:

  Implementieren eines Algorithmus, der das Traveling Tournament Problem effizient löst.
- Umsetzung:

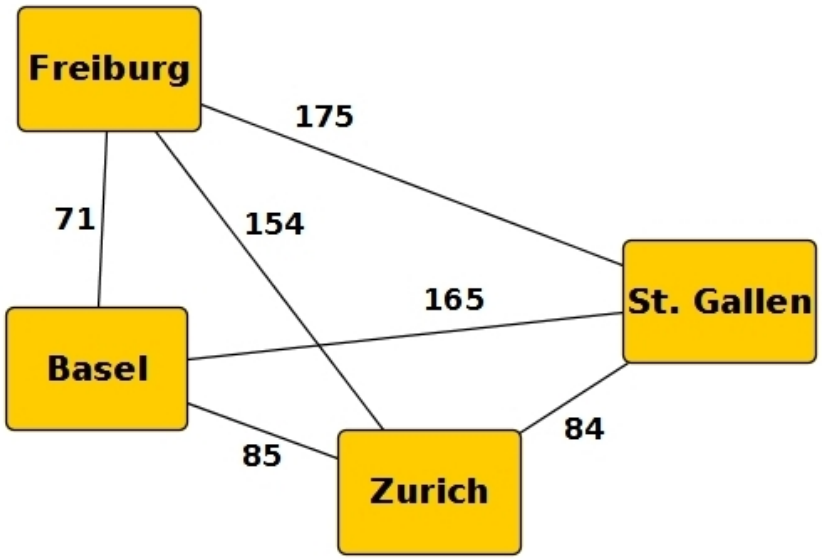
  mit einer Arbeit von Uthus et al. (2011) als Orientierungshilfe
- Programmiersprache:

  C++

Traveling Tournament Problem

Beispiel

Graphische Darstellung eines Traveling Tournament Problems



Ziel:

- finden eines Spielplans für ein doppeltes Rundenturnier
- Spielplan soll minimale Reisekosten verursachen

Problemkomplexität steigt exponentiel mit der Anzahl der Teams

zusätzliche Bedingungen

Es gibt zwei weitere Bedingungen an den Spielplan, die das Problem schwieriger machen:

- **at most constraint**: Es dürfen höchstens b aufeinanderfolgende Heim- bzw. Auswärtsspiele bestritten werden.
- **no repeat constraint**: Eine Paarung die an einem Spieltag stattfindet, darf nicht am darauf folgenden Spieltag wiederhohlt werden.

zusätzliche Bedingungen

Spielplan, der die *at most* Bedingung mit $b = 2$ verletzt

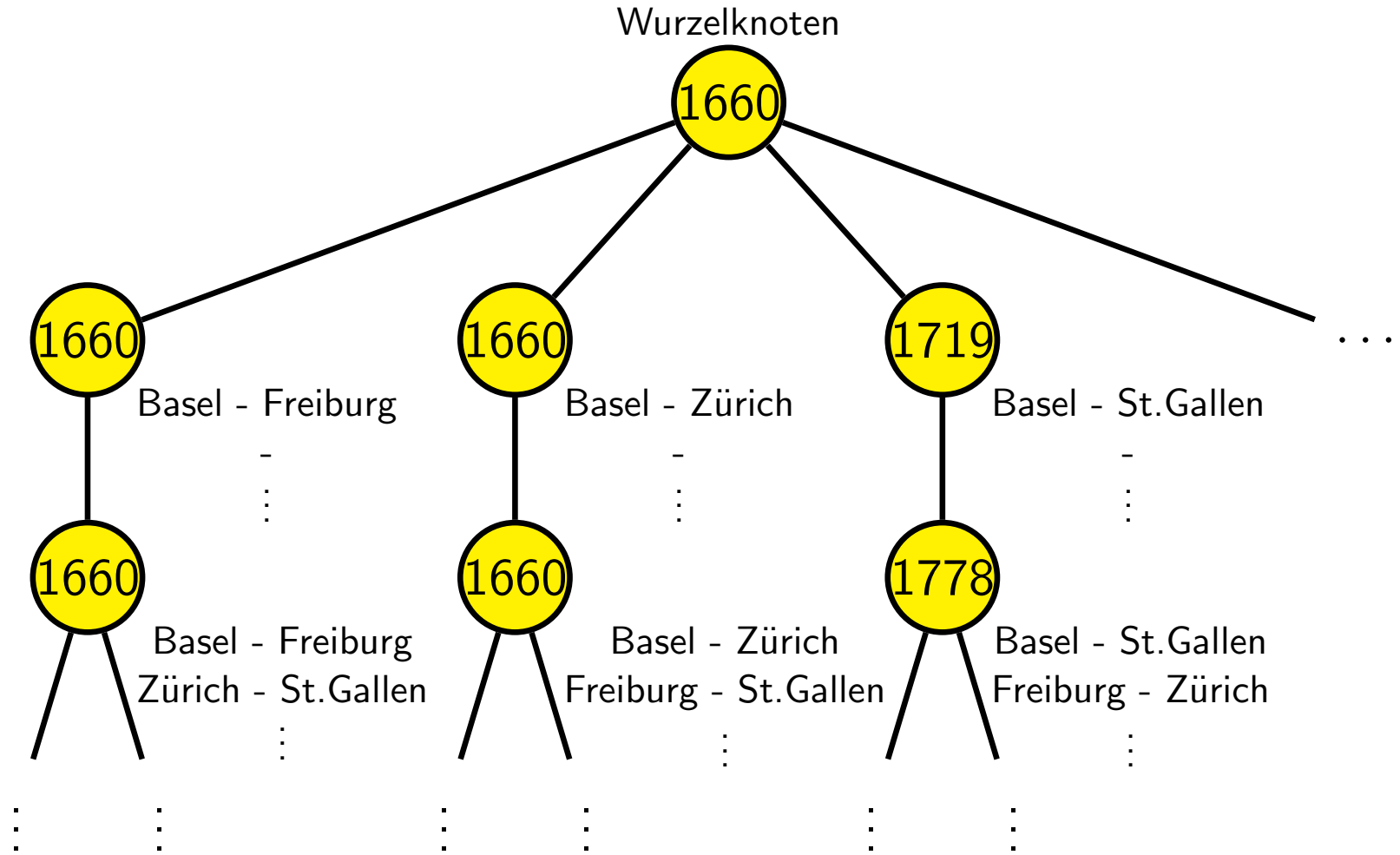
Matchday 1	Zurich	-	Basel
	St.Gallen	-	Freiburg
Matchday 2	Zurich	-	Freiburg
	St.Gallen	-	Basel
Matchday 3	Basel	-	Freiburg
	Zurich	-	St.Gallen
Matchday 4	Basel	-	Zurich
	Freiburg	-	St.Gallen
Matchday 5	Basel	-	St.Gallen
	Freiburg	-	Zurich
Matchday 6	Freiburg	-	Basel
	St.Gallen	-	Zurich

zusätzliche Bedingungen

Spielplan, der die *no repeat* Bedingung verletzt

Matchday 1	**Zurich**	**-**	**Basel**
	St.Gallen	**-**	**Freiburg**
Matchday 2	**Zurich**	**-**	**Freiburg**
	St.Gallen	**-**	**Basel**
Matchday 3	**Basel**	**-**	**Freiburg**
	Zurich	**-**	**St.Gallen**
Matchday 4	**Freiburg**	**-**	**Basel**
	St.Gallen	**-**	**Zurich**
Matchday 5	**Basel**	**-**	**St.Gallen**
	Freiburg	**-**	**Zurich**
Matchday 6	**Basel**	**-**	**Zurich**
	Freiburg	**-**	**St.Gallen**

Suchraum



Teamreihenfolge = <Basel, Freiburg, Zürich, St.Gallen >

Heuristik

ILB Heuristik

Um die f-Werte der einzelnen Knoten im Suchbaum zu bestimmen, habe ich die **Independent Lower Bound** Heuristik verwendet:

- berechnet untere Schranke $\mathbf{bound_{tot}}$ für totale Reisekosten, die in Lösungen unterhalb des betrachteten Knotens entstehen können:

$$\mathbf{bound_{tot}} = \sum_{\mathbf{t \in T}} \mathbf{bound_t}$$

- die unteren Schranken für die einzelnen Teams werden unabhängig von einander berechnet:
  1. generiere alle möglichen Auswärtsreisen
  2. berechne deren Kosten
  3. suche günstigste Kombination von Auswärtsreisen
- Heuristik ist **zulässig** und **monoton**

Suchverfahren

Standard IDA* Suche

- IDA* Suche ist ein weit verbreiteter Ansatz um klassische Suchprobleme zu lösen.
- IDA* ist eine iterative Tiefensuche.
- Sie kombiniert:

  die Vorteile einer Breitensuche:

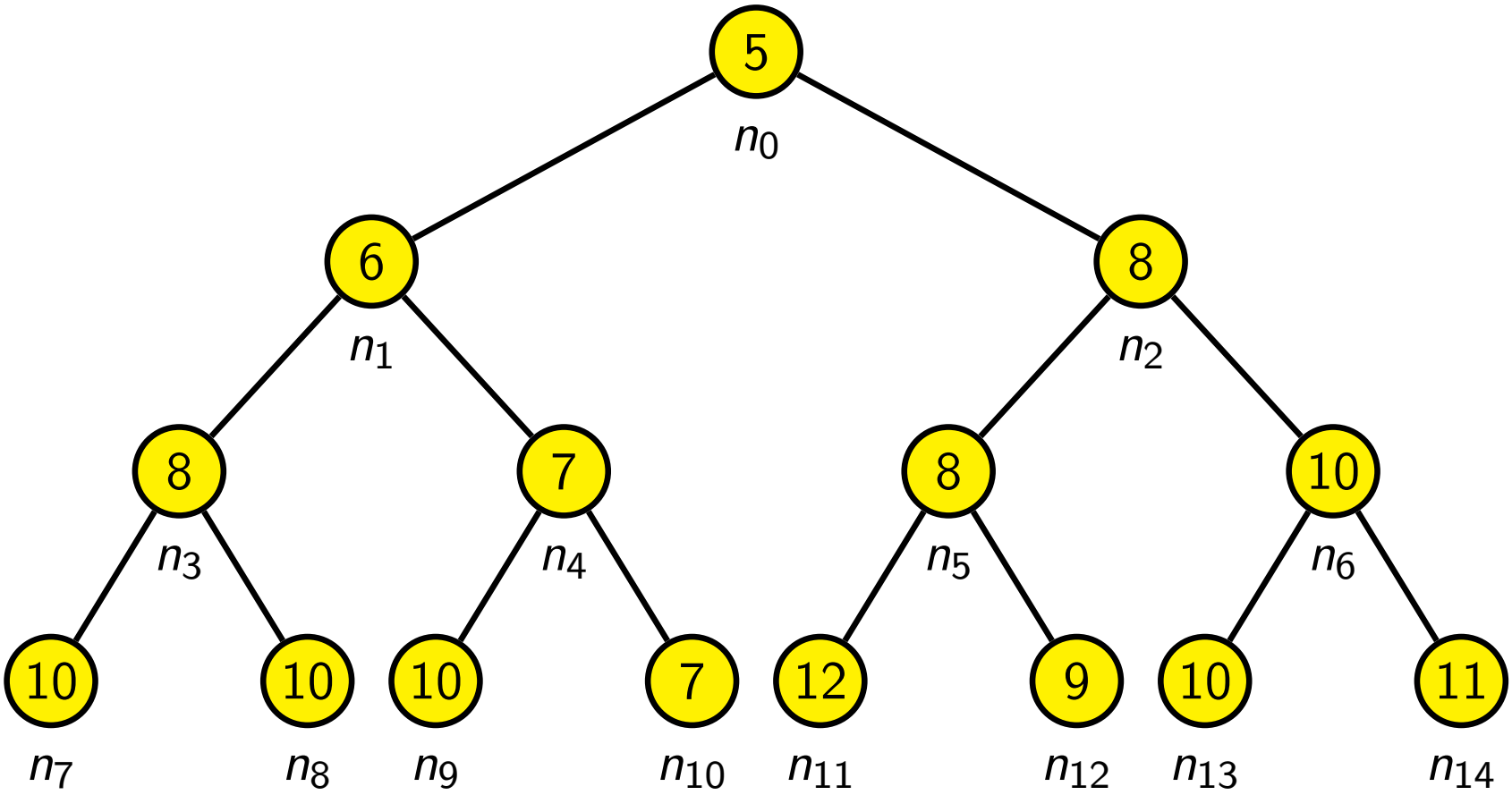
  Optimalitätsgarantie

  mit den Vorteilen einer Tiefensuche:

  wenig Speicherbedarf

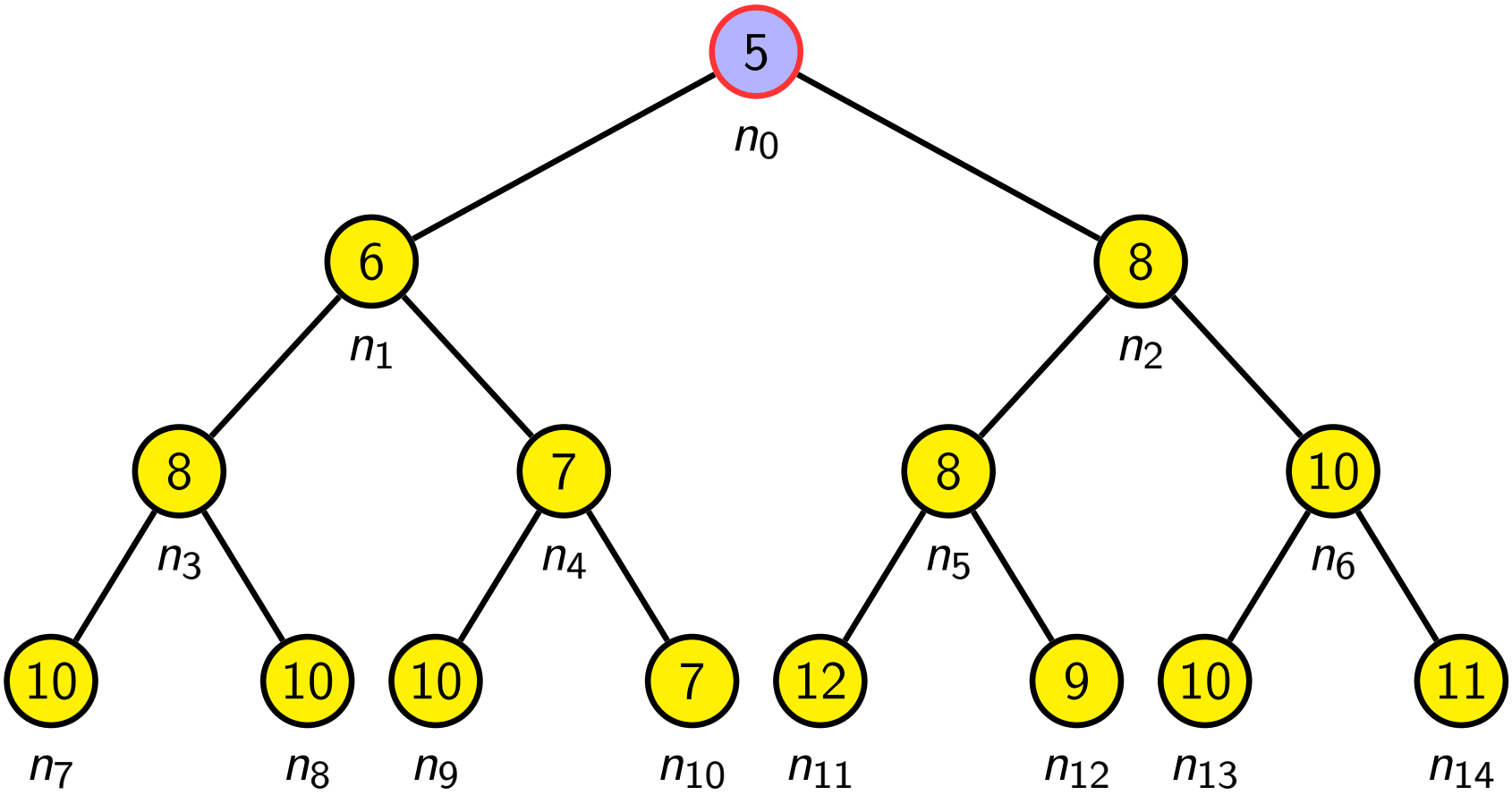
Standard IDA* Suche

1. Iteration: $\mathbf{f_{cur} = 0, f_{next} = \infty}$



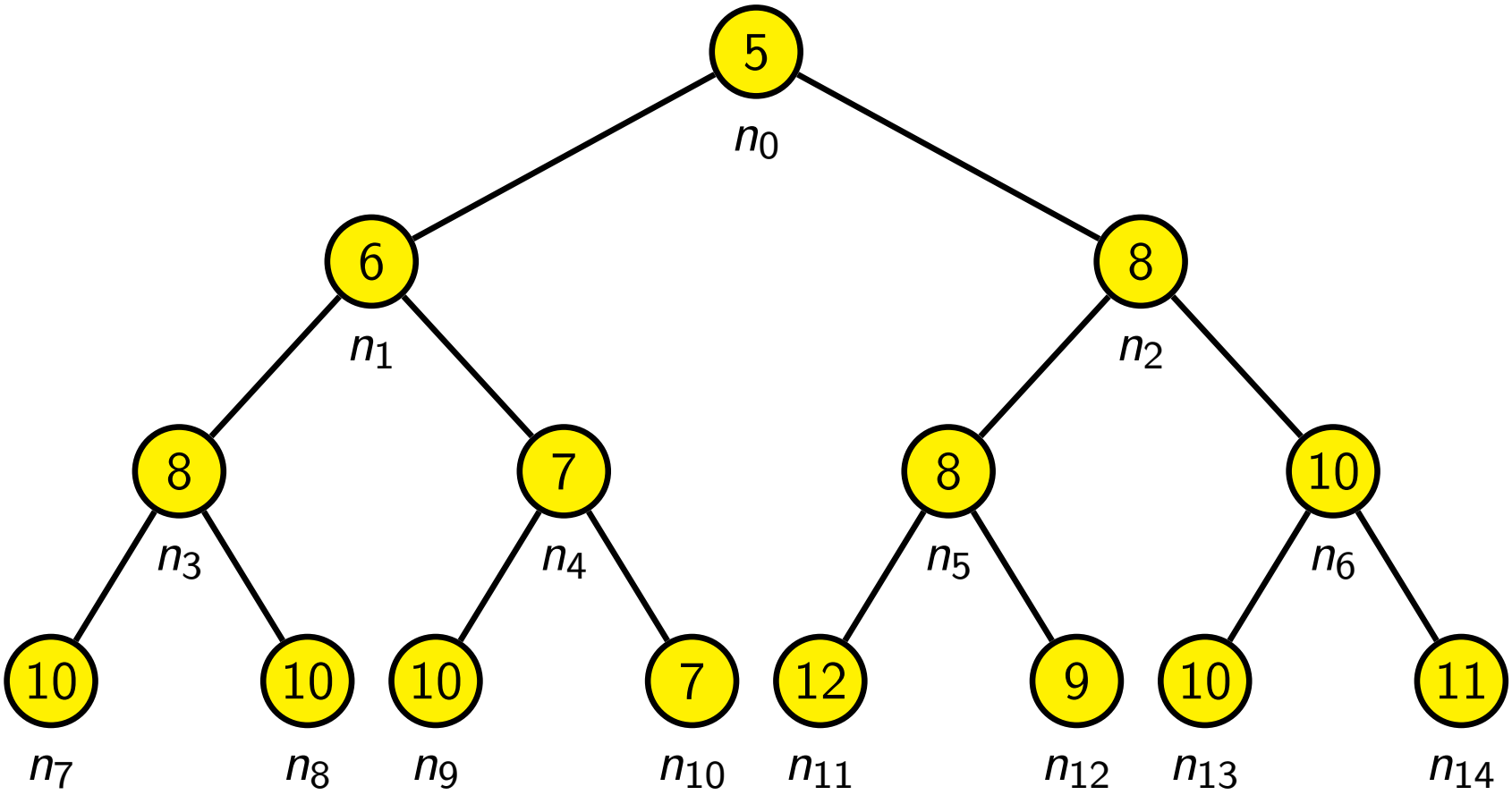
Standard IDA* Suche

1. Iteration: $\mathbf{f_{cur} = 0, f_{next} = 5}$



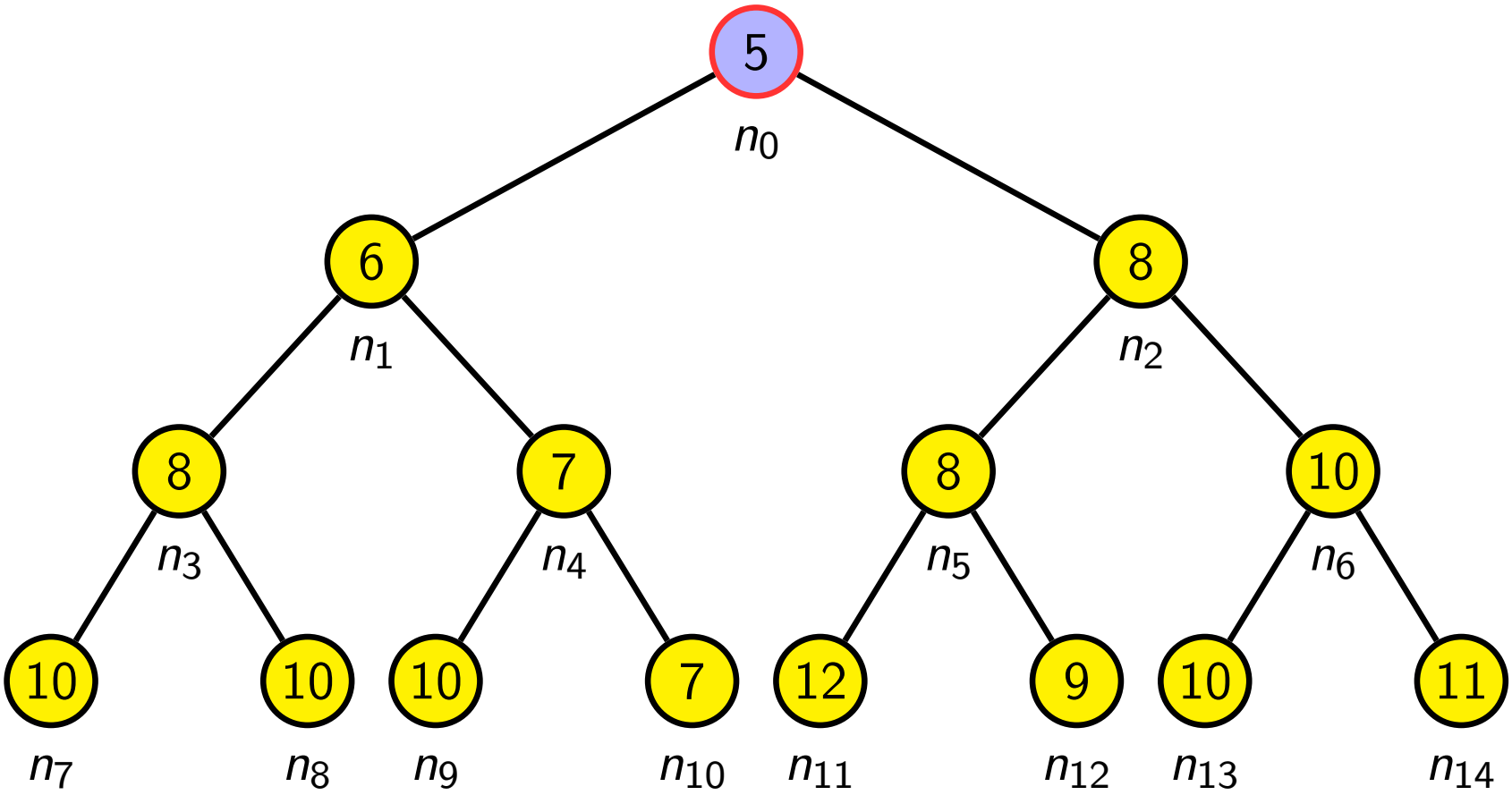
Standard IDA* Suche

2. Iteration: $\mathbf{f_{cur} = 5, f_{next} = \infty}$



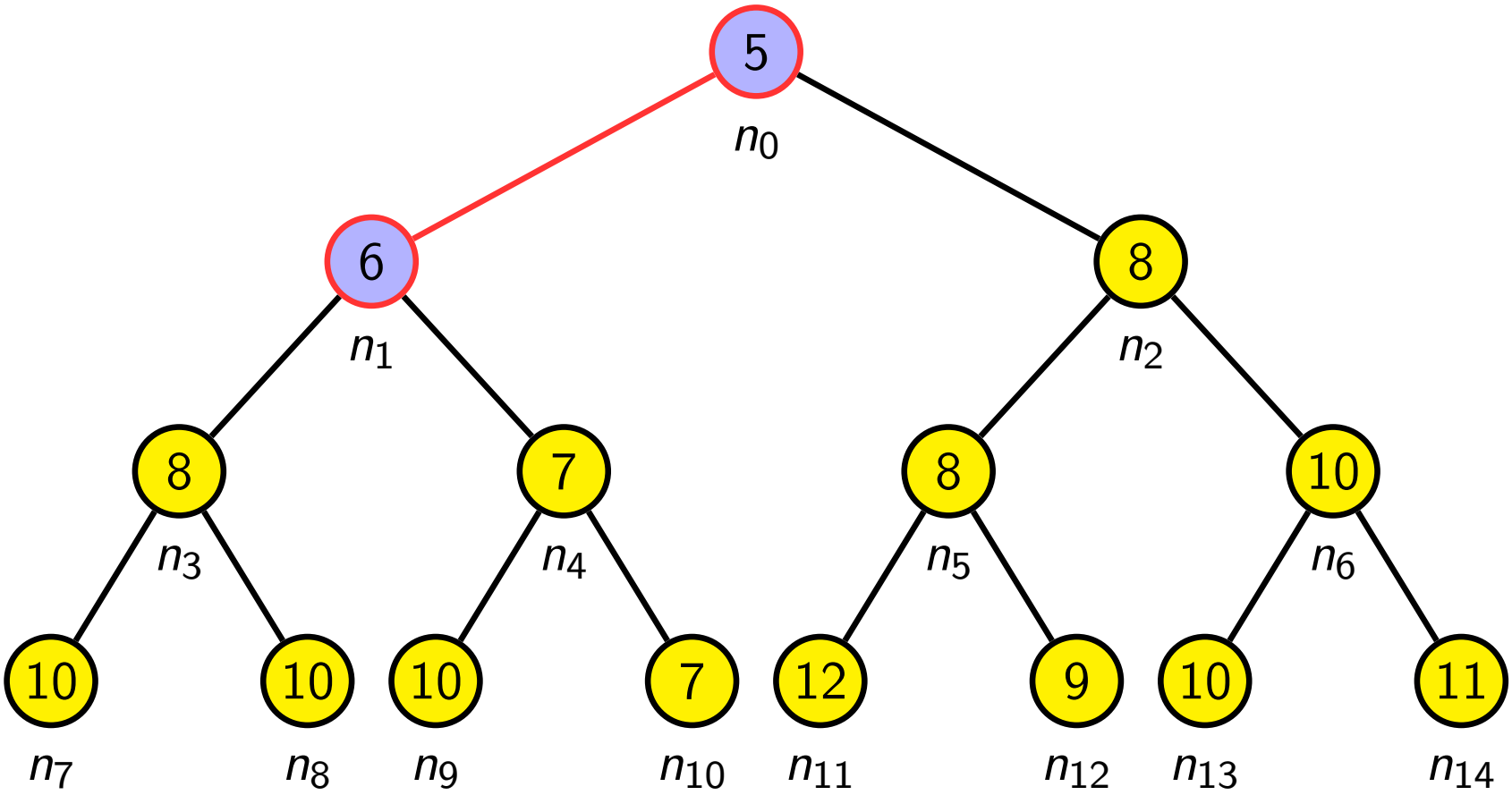
Standard IDA* Suche

2. Iteration: $\mathbf{f_{cur} = 5, f_{next} = \infty}$



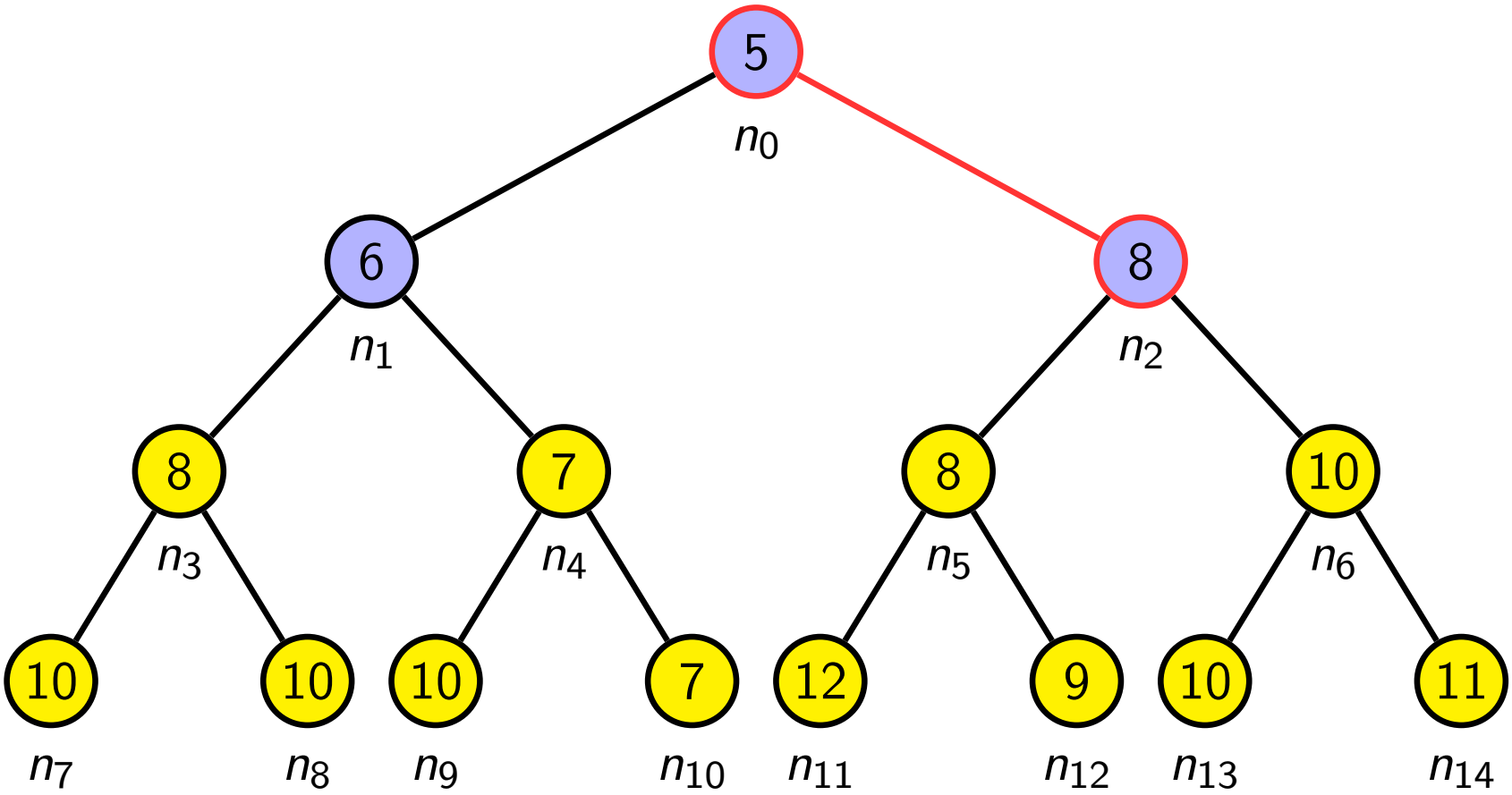
Standard IDA* Suche

2. Iteration: $\mathbf{f_{cur} = 5, f_{next} = 6}$



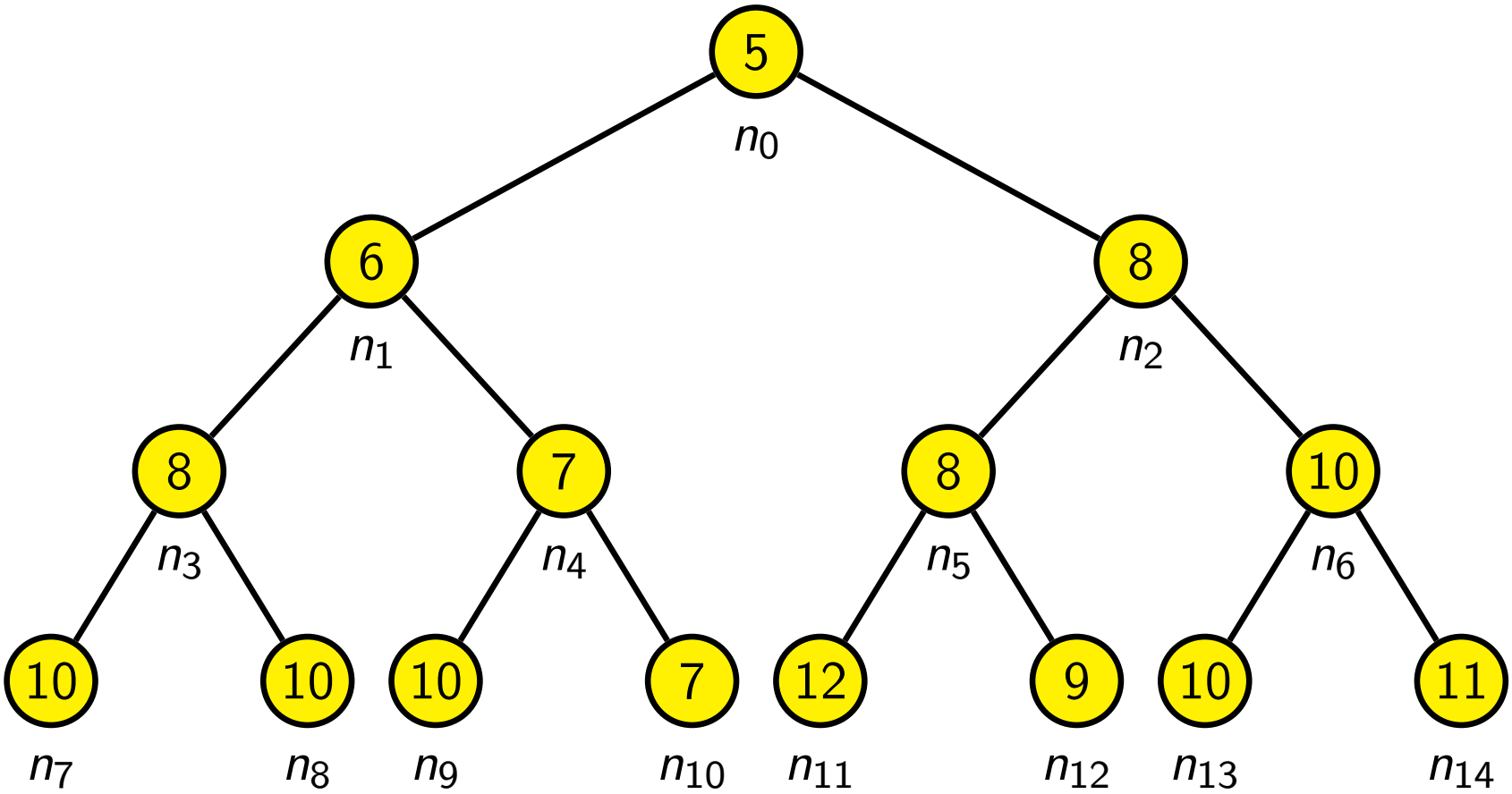
Standard IDA* Suche

2. Iteration: $\mathbf{f_{cur} = 5, f_{next} = 6}$



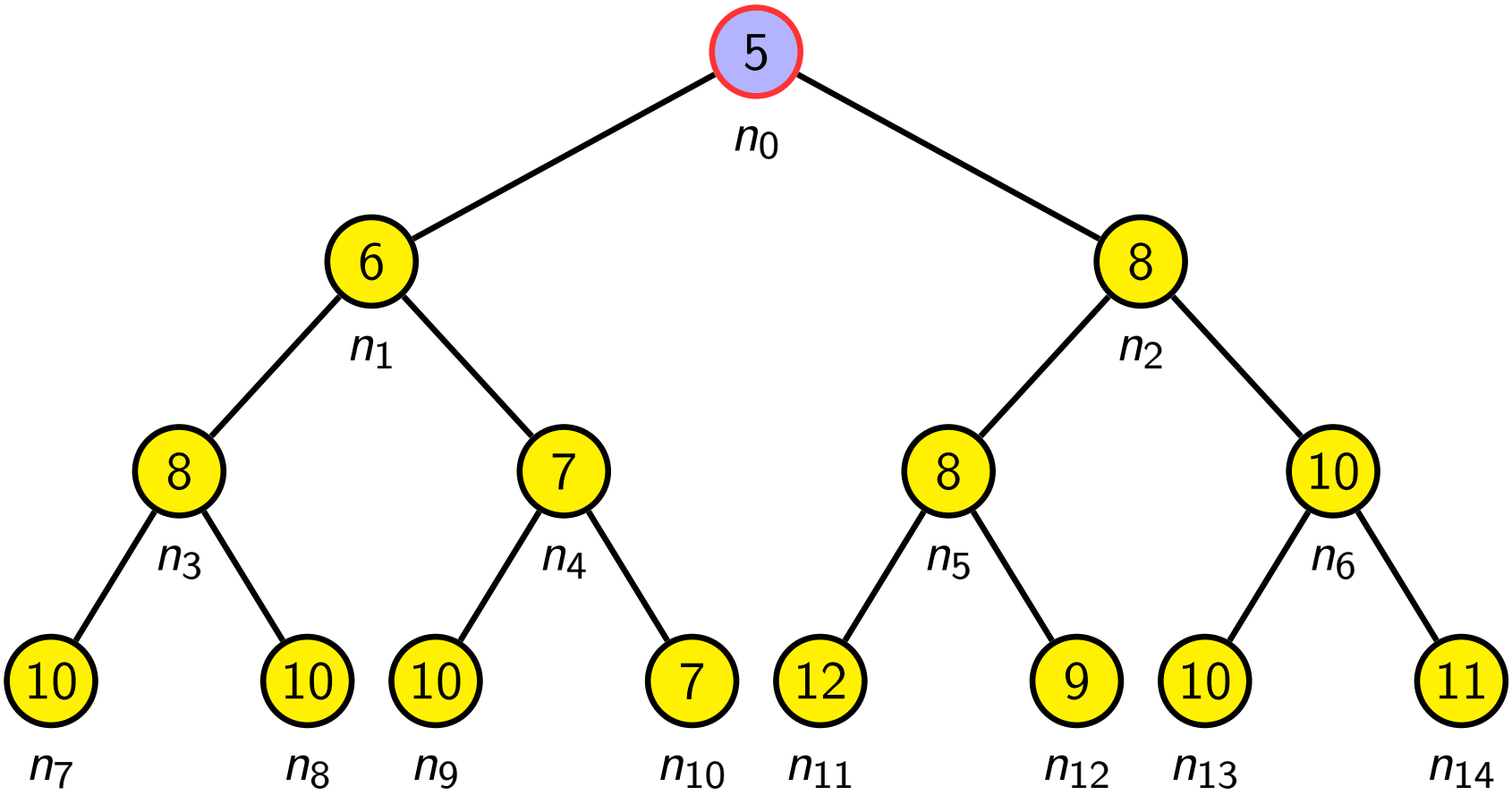
Standard IDA* Suche

3. Iteration: $\mathbf{f_{cur} = 6, f_{next} = \infty}$



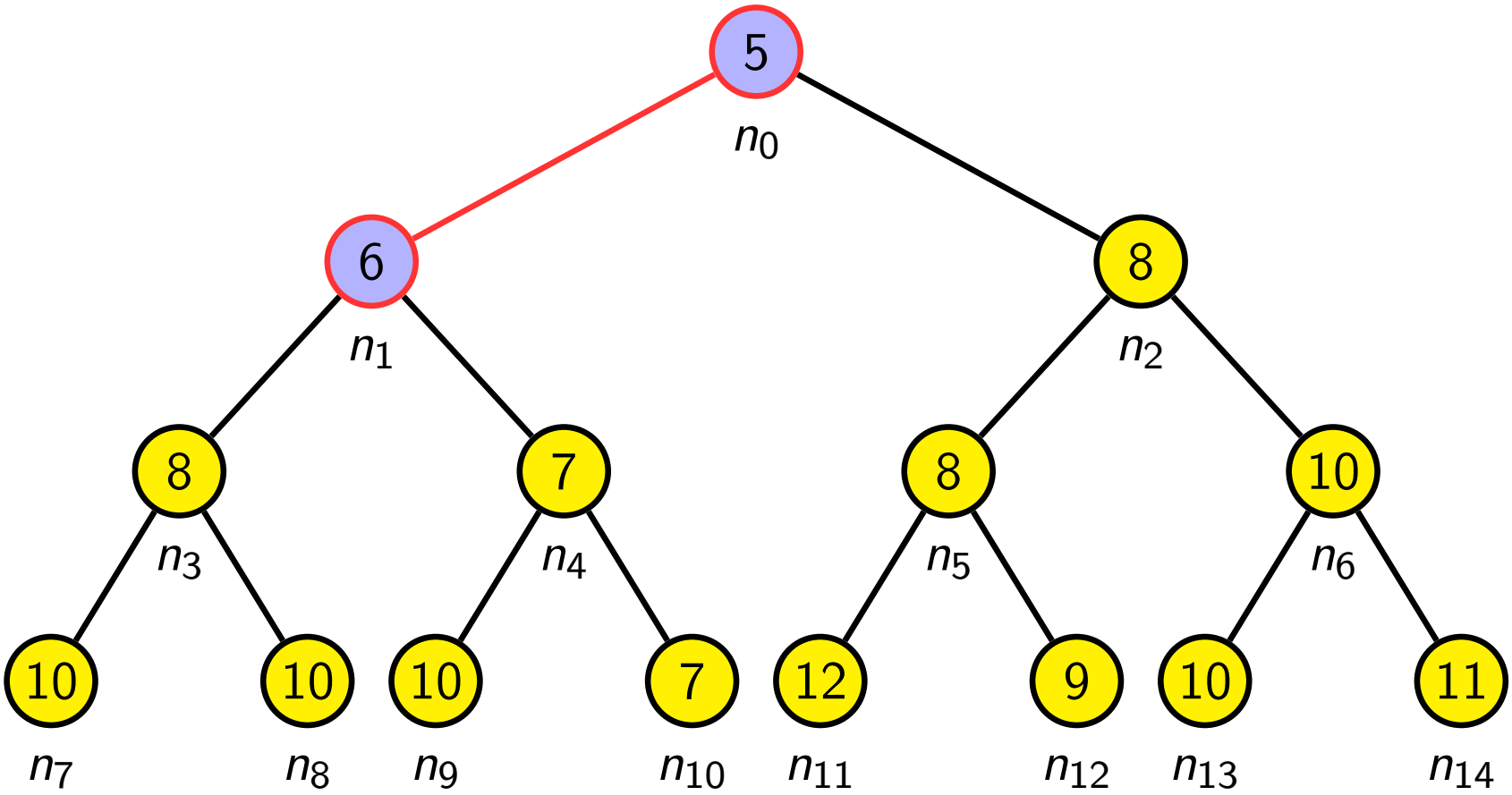
Standard IDA* Suche

3. Iteration: $\mathbf{f_{cur} = 6, f_{next} = \infty}$



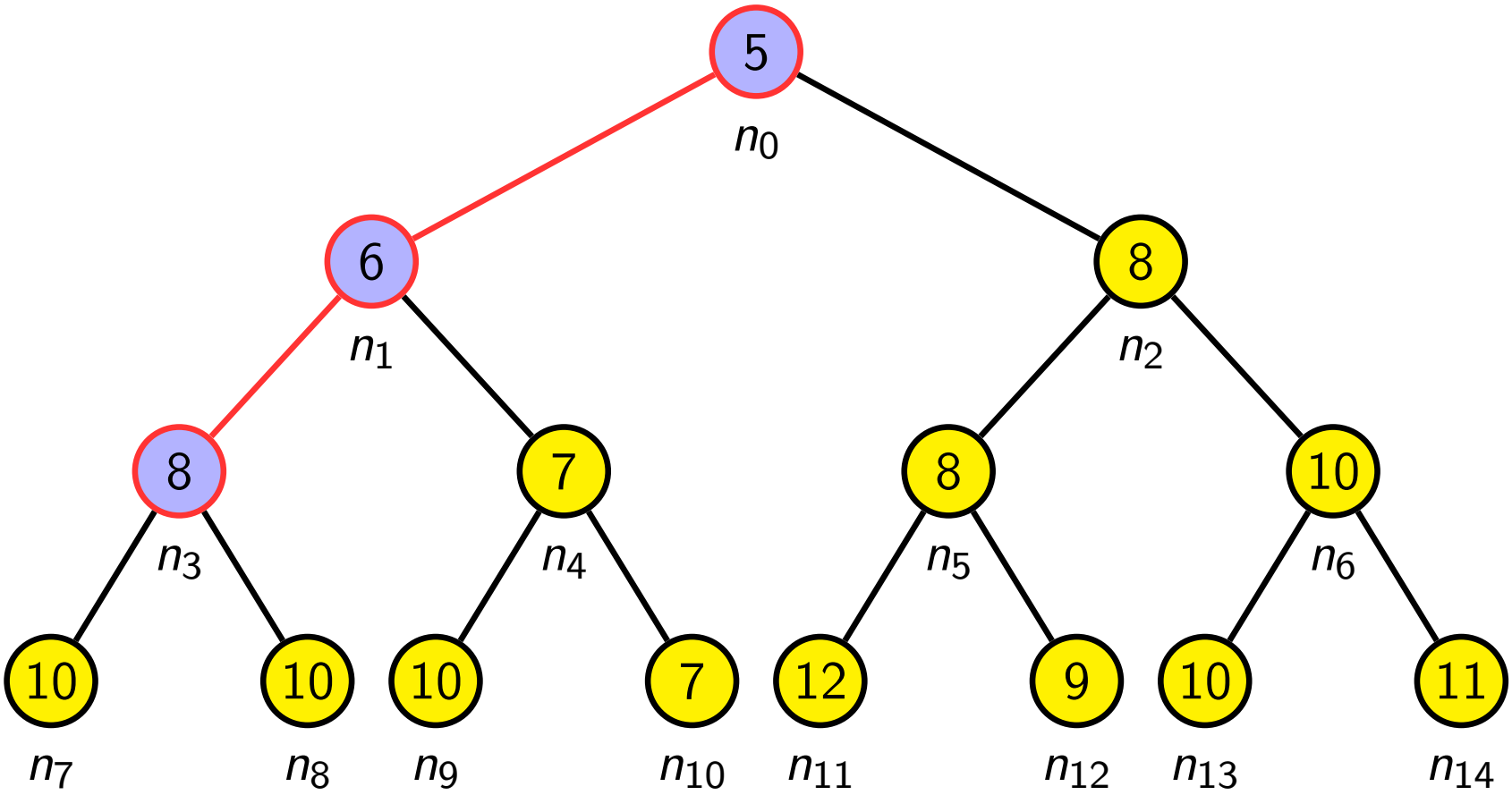
Standard IDA* Suche

3. Iteration: $\mathbf{f_{cur} = 6, f_{next} = \infty}$



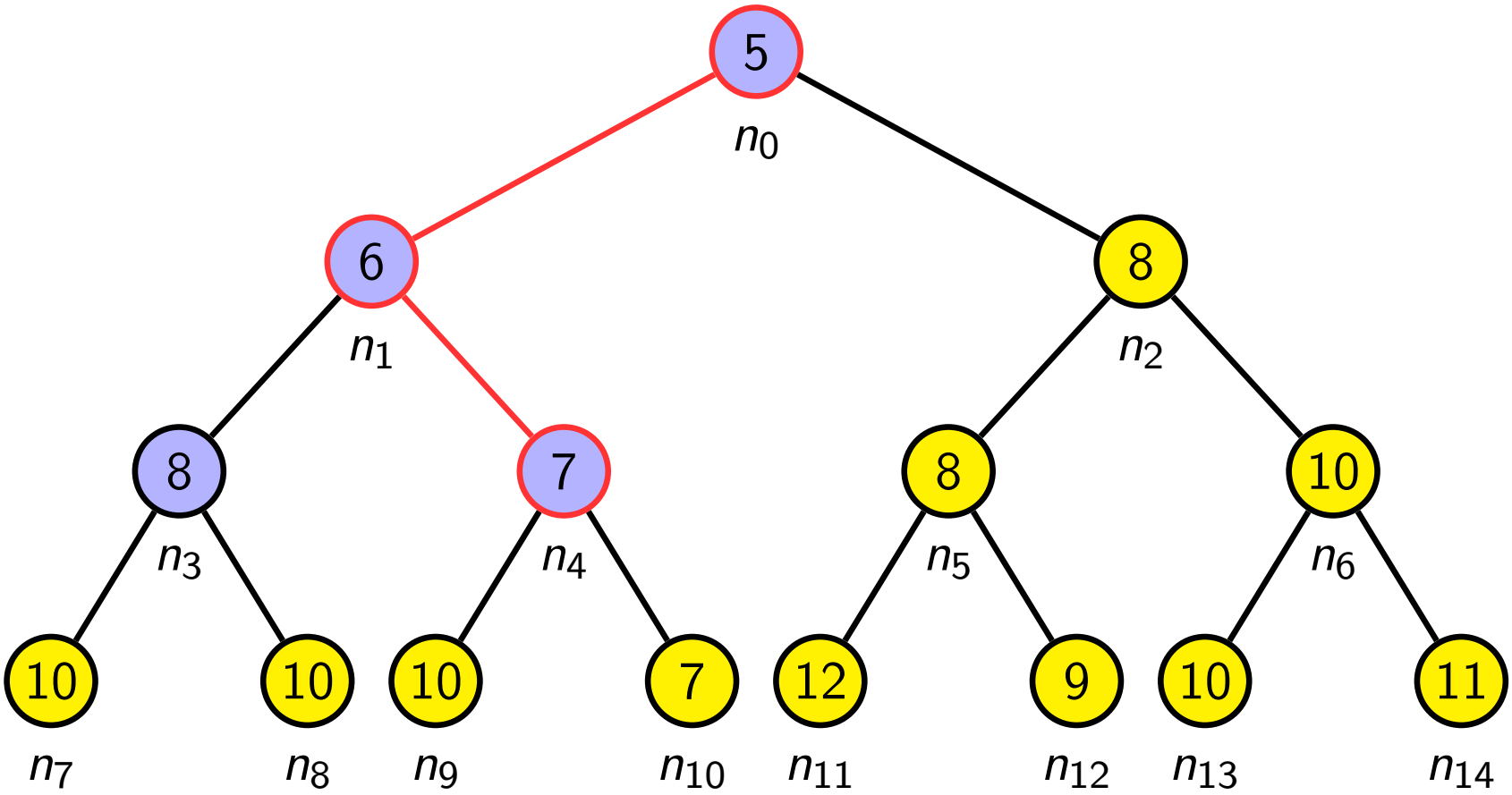
Standard IDA* Suche

3. Iteration: $\mathbf{f_{cur} = 6, f_{next} = 8}$



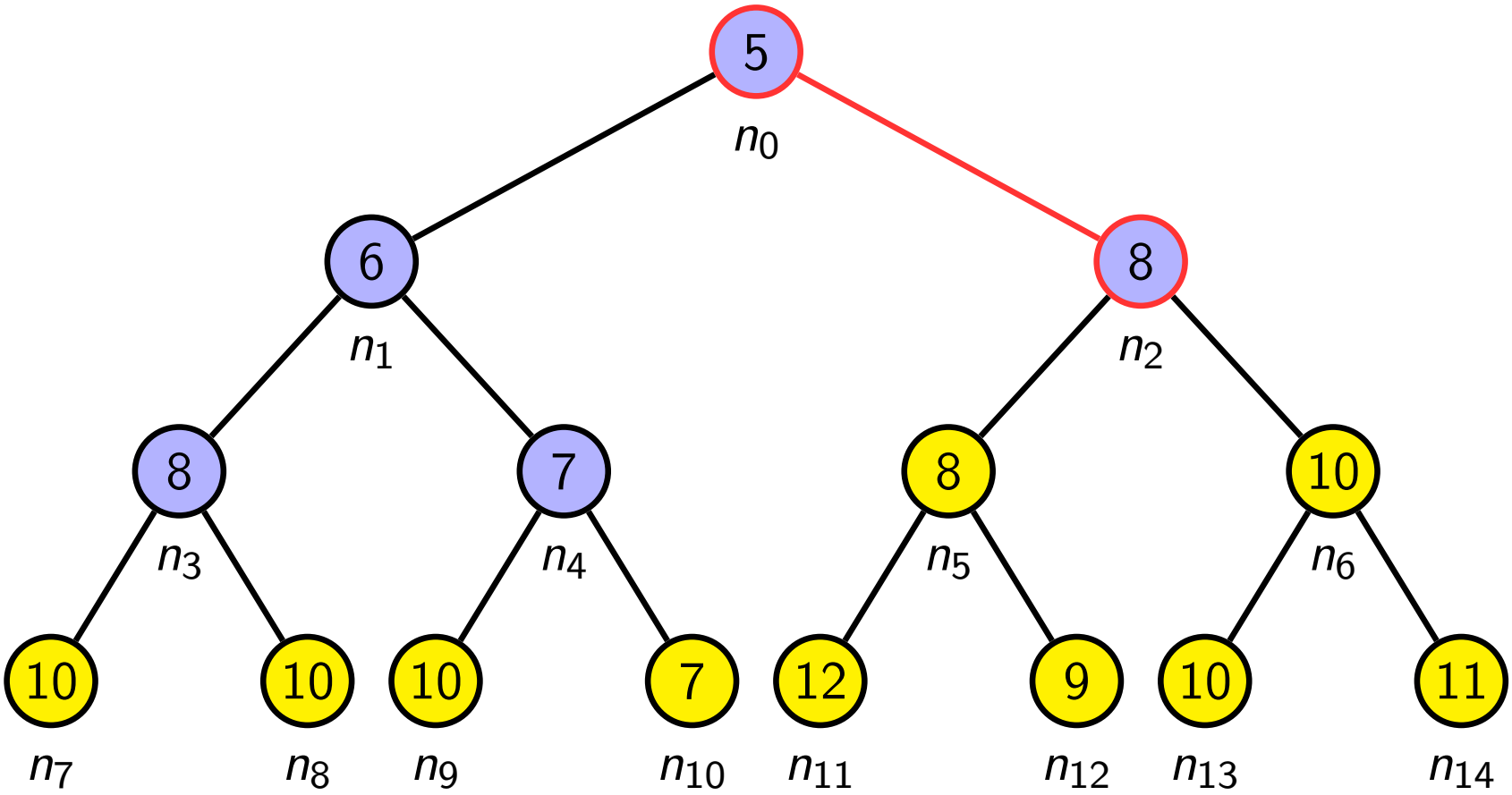
Standard IDA* Suche

3. Iteration: $\mathbf{f_{cur} = 6, f_{next} = 7}$



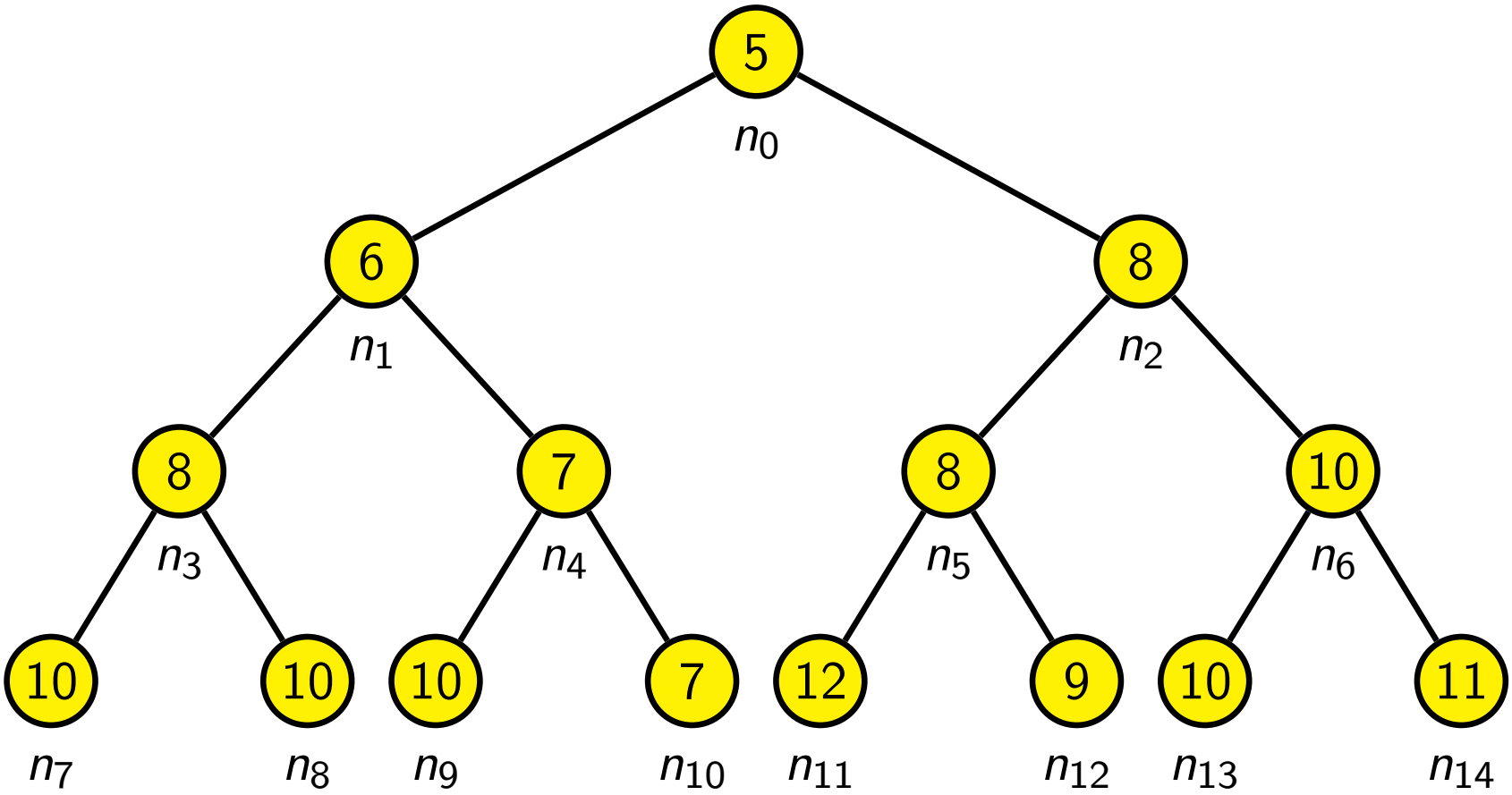
Standard IDA* Suche

3. Iteration: $\mathbf{f_{cur} = 6, f_{next} = 7}$



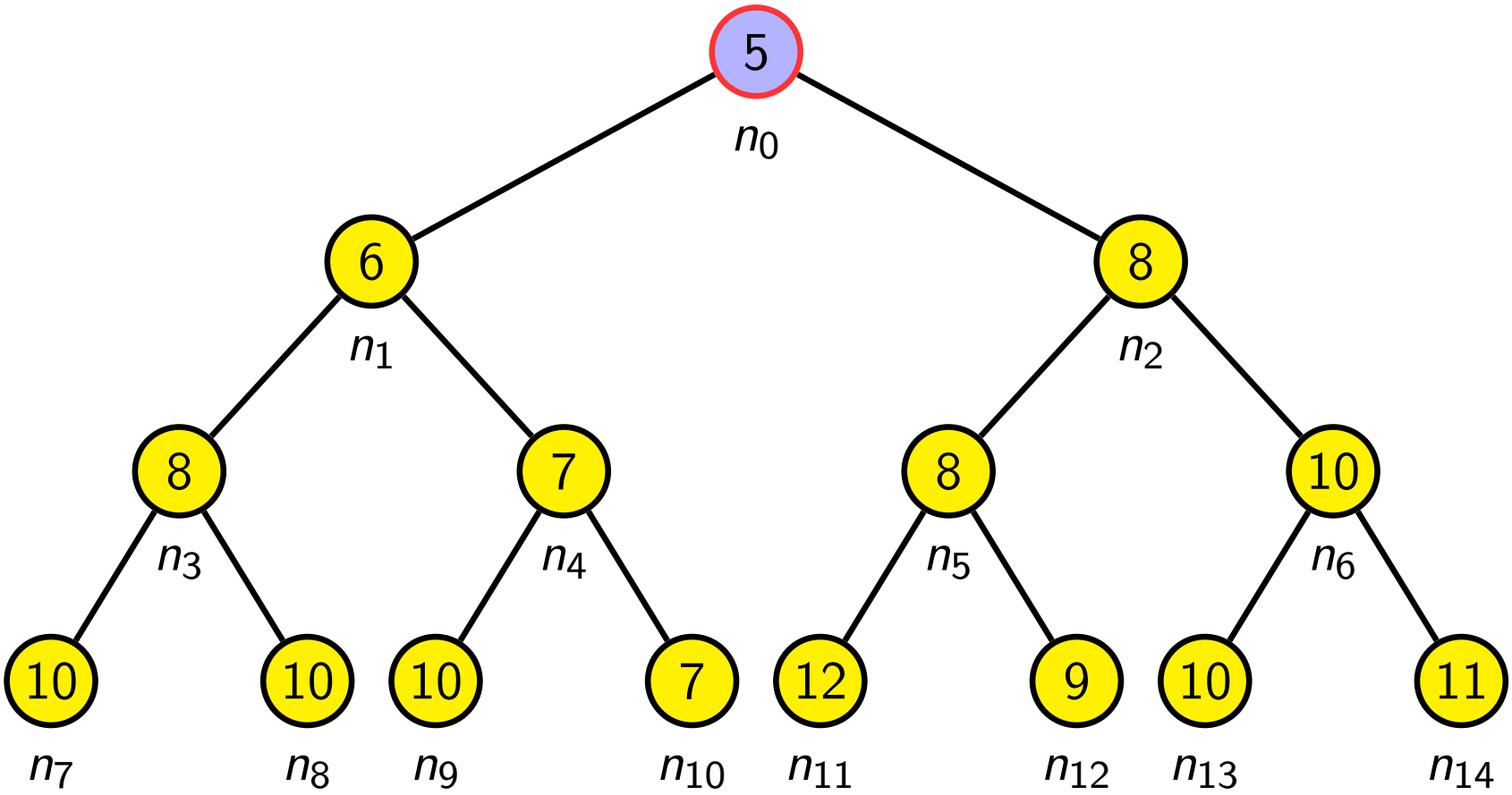
Standard IDA* Suche

4. Iteration: $\mathbf{f_{cur} = 7, f_{next} = \infty}$



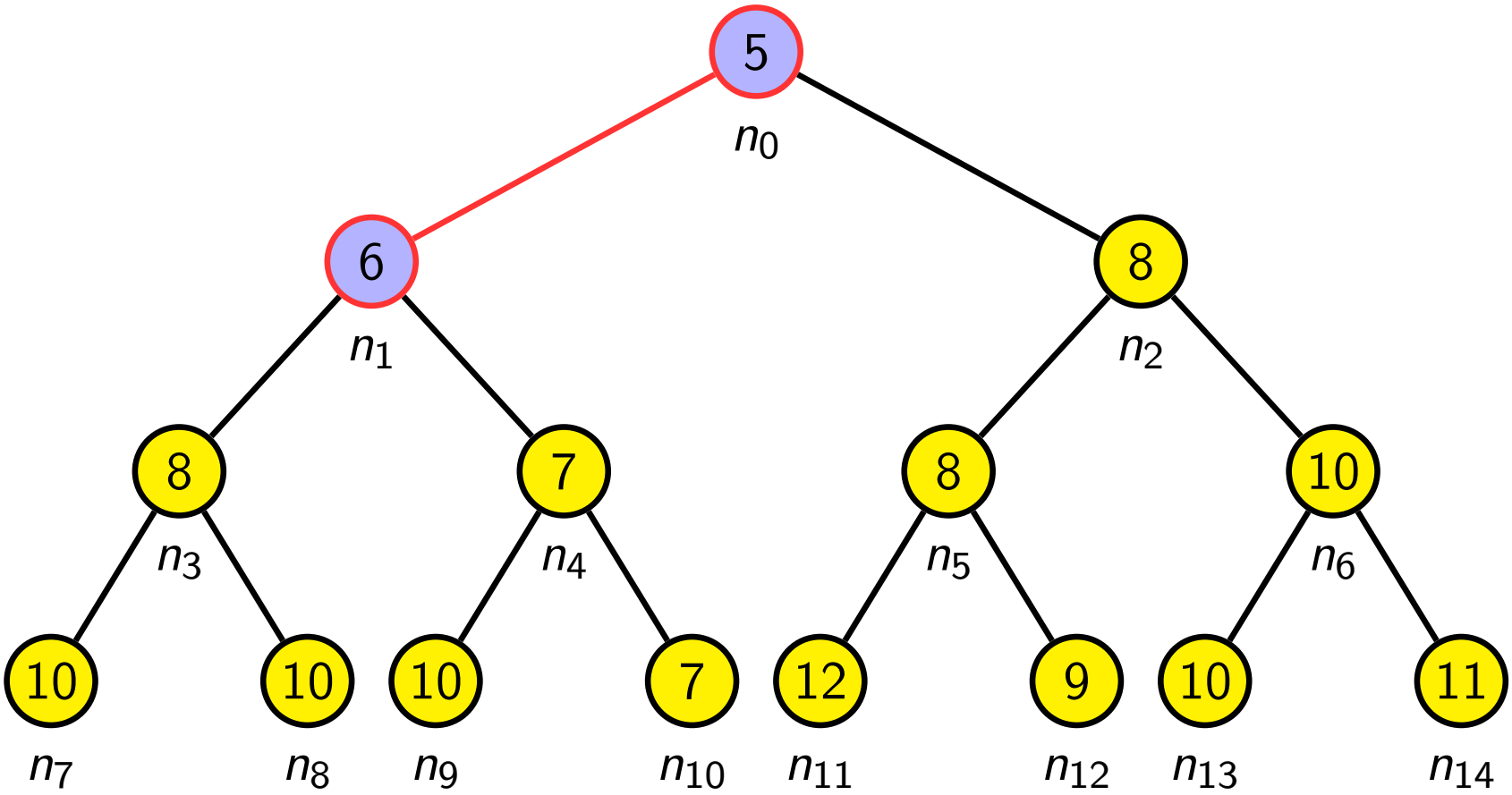
Standard IDA* Suche

4. Iteration: $\mathbf{f_{cur} = 7, f_{next} = \infty}$



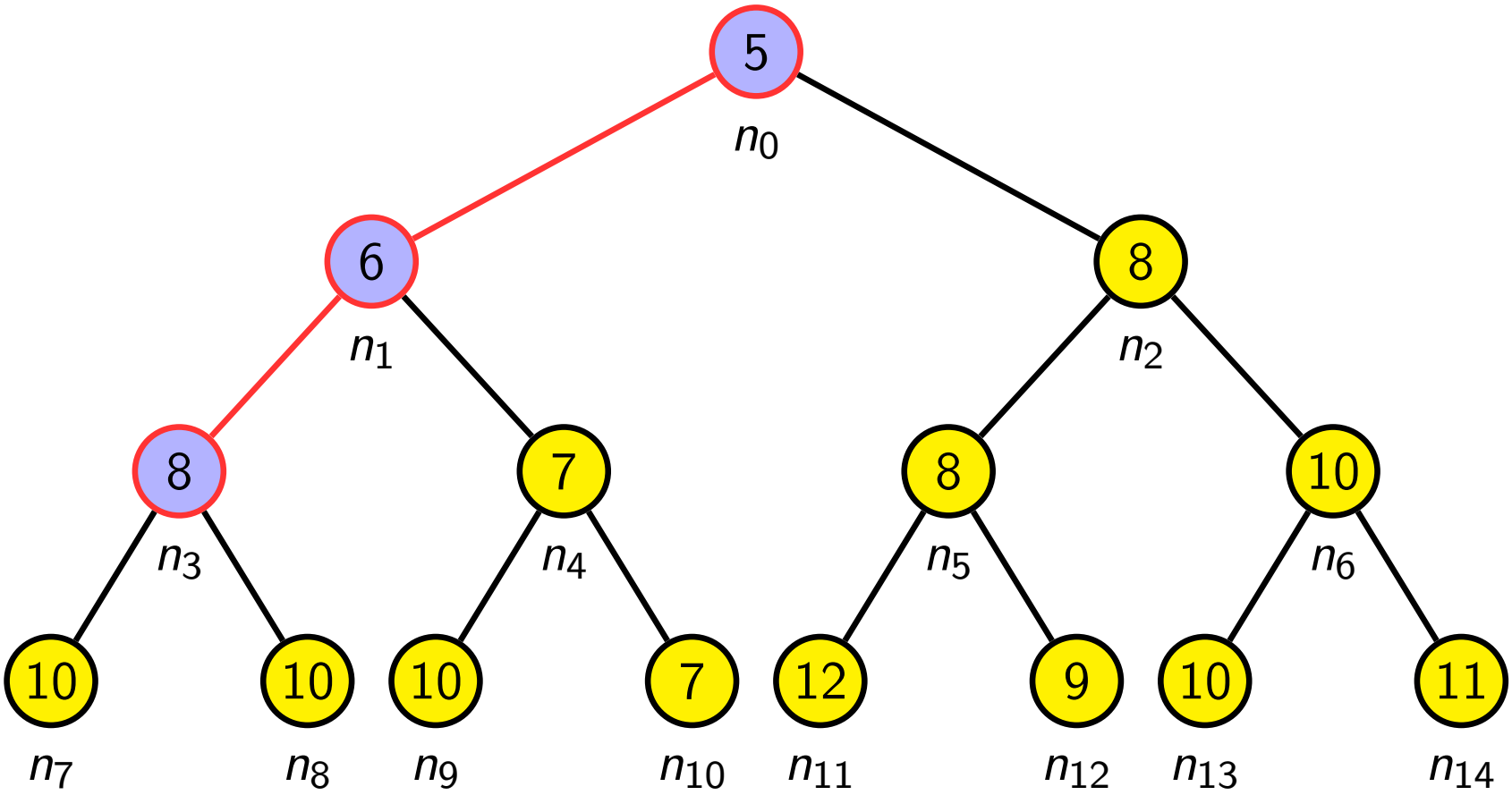
Standard IDA* Suche

4. Iteration: $\mathbf{f_{cur} = 7, f_{next} = \infty}$



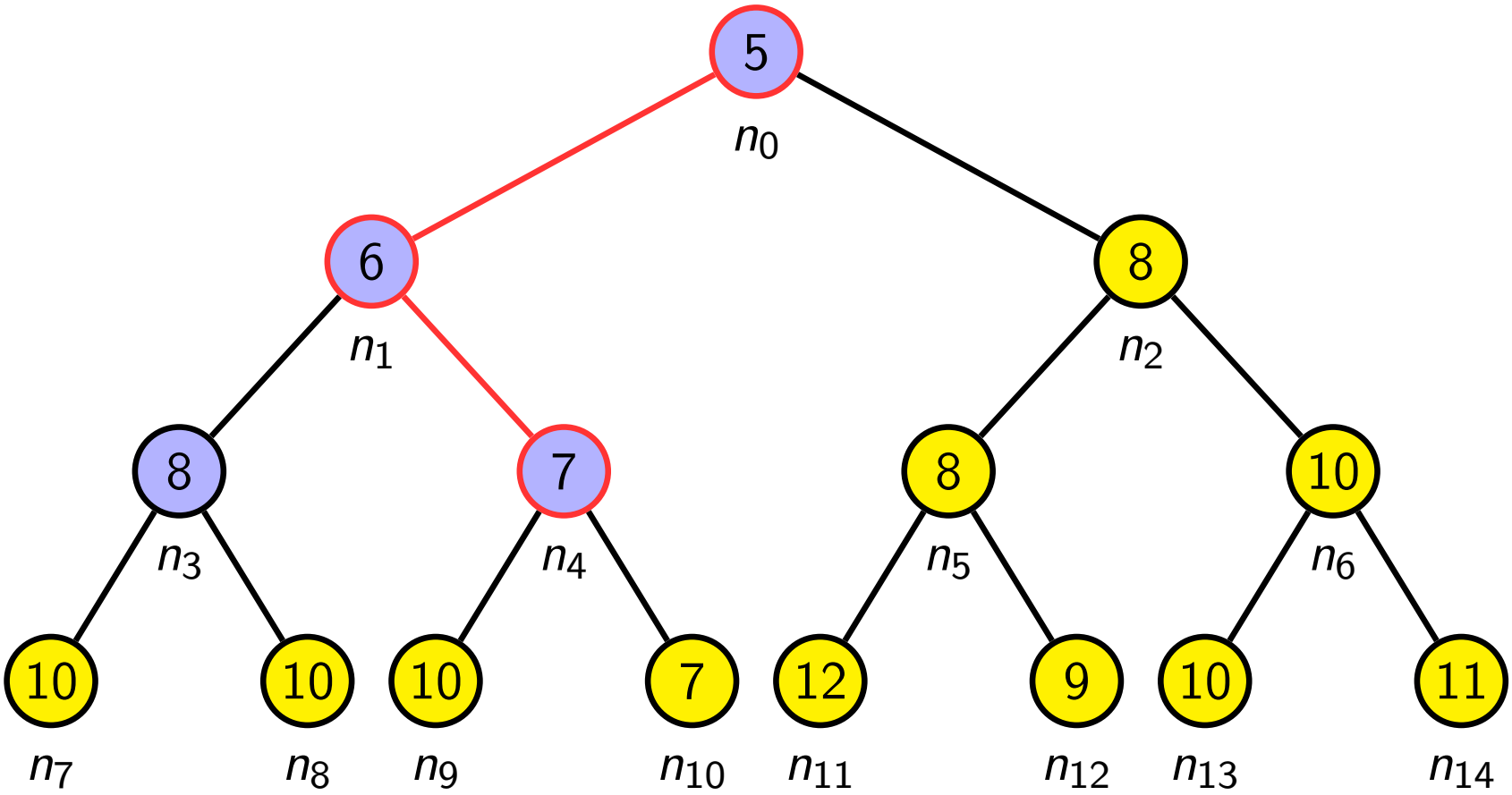
Standard IDA* Suche

4. Iteration: $\mathbf{f_{cur} = 7, f_{next} = 8}$



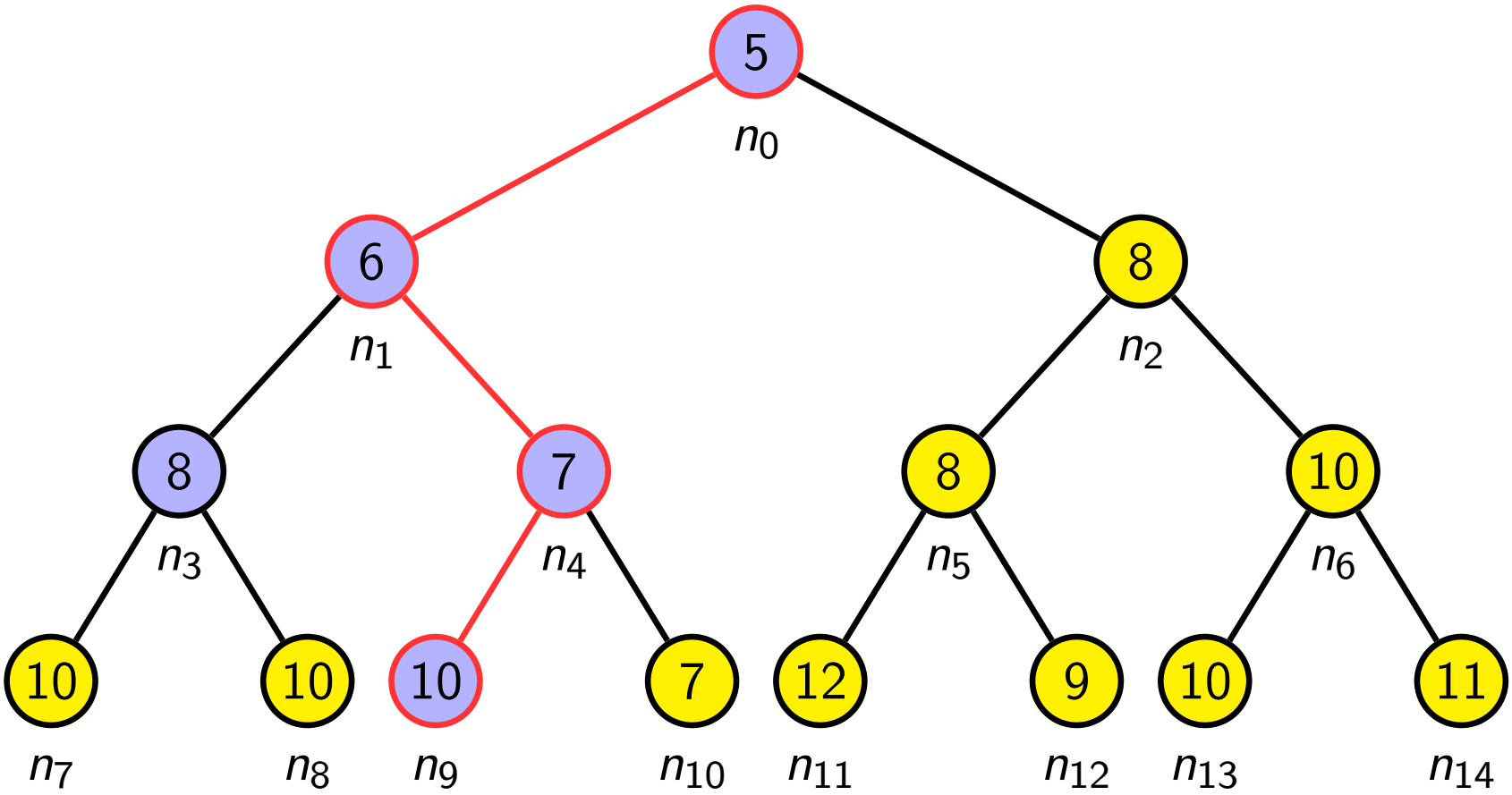
Standard IDA* Suche

4. Iteration: $\mathbf{f_{cur} = 7, f_{next} = 8}$



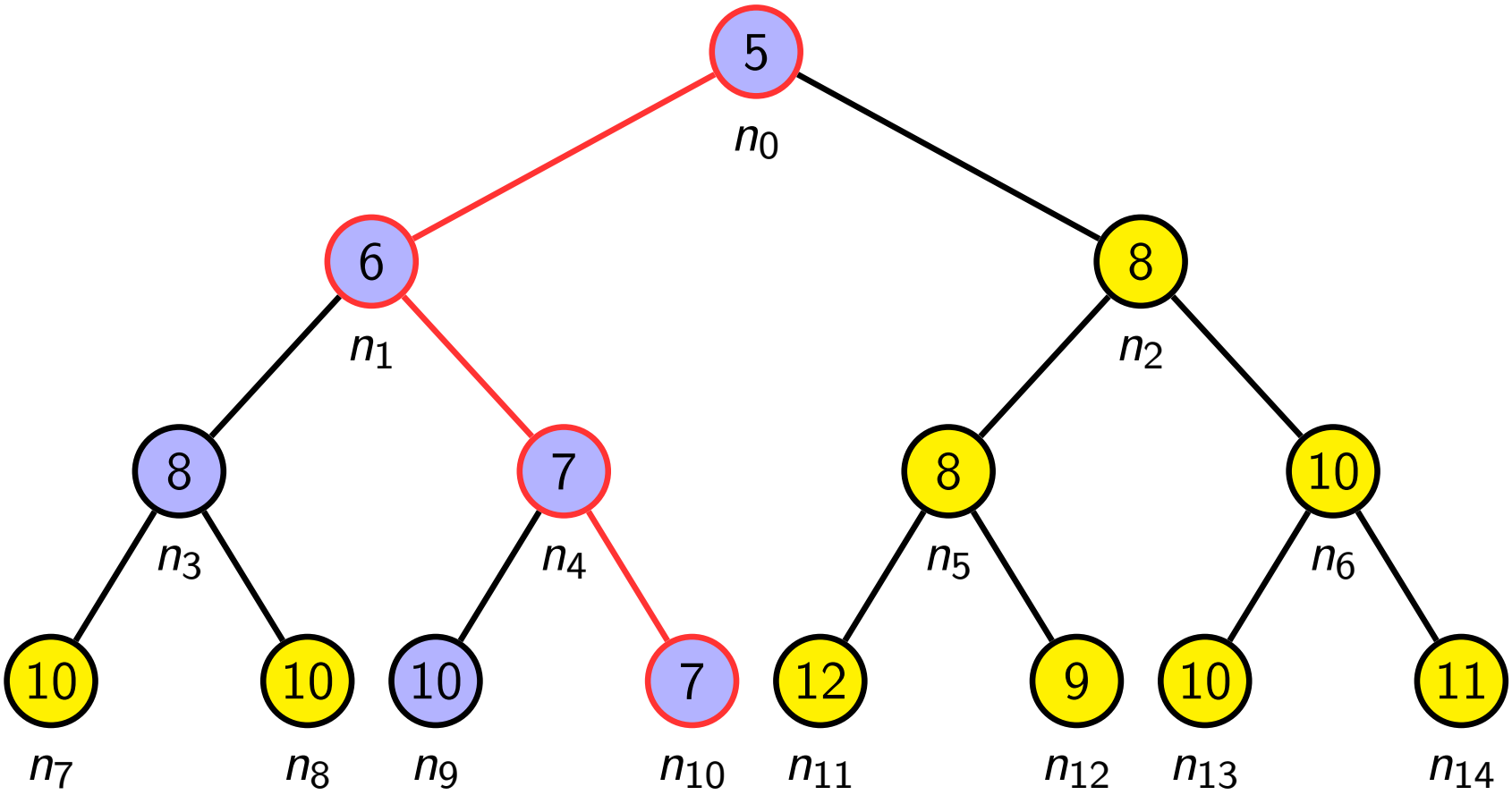
Standard IDA* Suche

4. Iteration: $\mathbf{f_{cur} = 7, f_{next} = 8}$



Standard IDA* Suche

4. Iteration: $\mathbf{f_{cur} = 7, f_{next} = 8}$



Erweiterungen der IDA* Suche

Uthus et al. (2011) schlagen 3 grundsätzliche Erweiterungen der IDA* Suche vor:

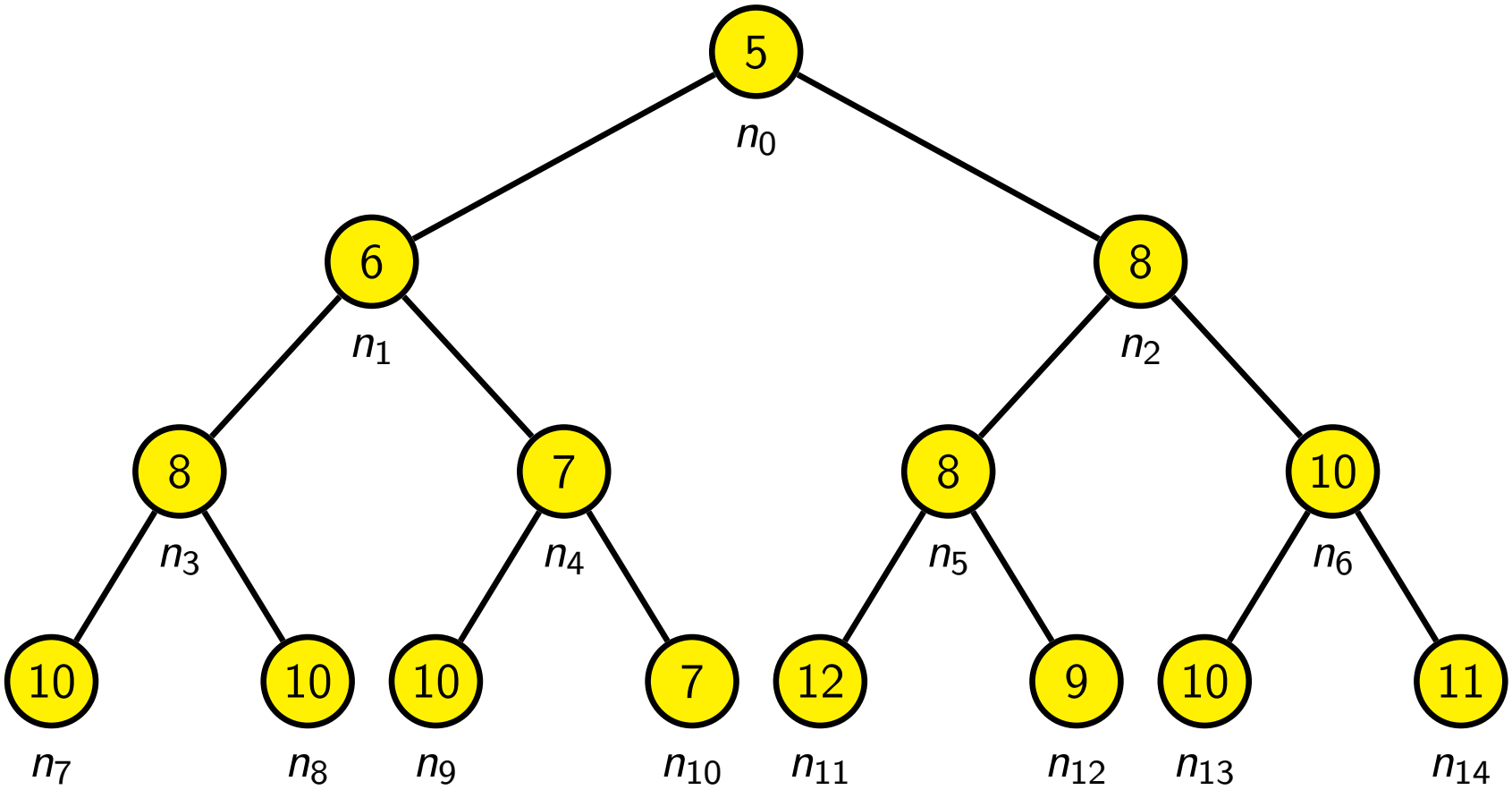
- **Forced Deepening**: zwinge die Suche mit jeder Iteration λ Levels tiefer in den Baum zu gehen
- **Elite Paths**: speichere in jeder Iteration den Pfad zum Knoten, bei dem der f_{next}-Wert gefunden wurde und gehe zu Beginn einer Iteration zuerst dem gespeicherten Pfad der vorherigen Iteration entlang
- **Subtree Forests**: teile den Suchbaum in mehrere Teilbäume auf, die alle ein eigenes f-Limit verwalten
  $\Rightarrow$ in jeder Iteration können:
  - Teilbäume in günstiger Ordnung durchlaufen werden
  - manche Teilbäum nicht betrachtet werden

IDA* Suche mit Forced Deepening ($\lambda = 2$) und Elite Paths

IDA* Suche mit Forced Deepening ($\lambda = 2$) und Elite Paths

1. Iteration:

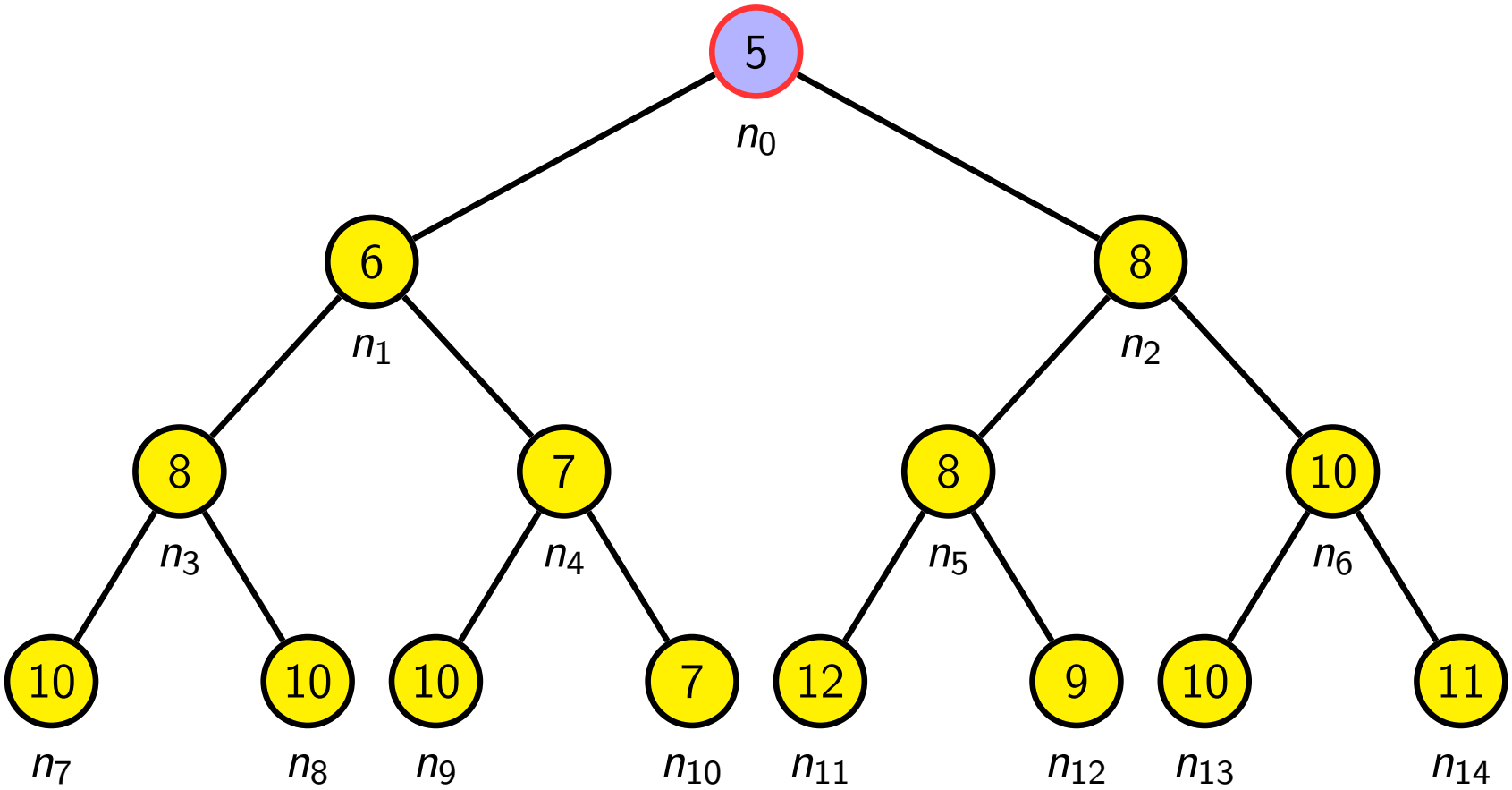
$\mathbf{f_{cur} = 0, f_{next} = \infty, ep_{cur} = <>, ep_{next} = <>}$



IDA* Suche mit Forced Deepening ($\lambda = 2$) und Elite Paths

1. Iteration:

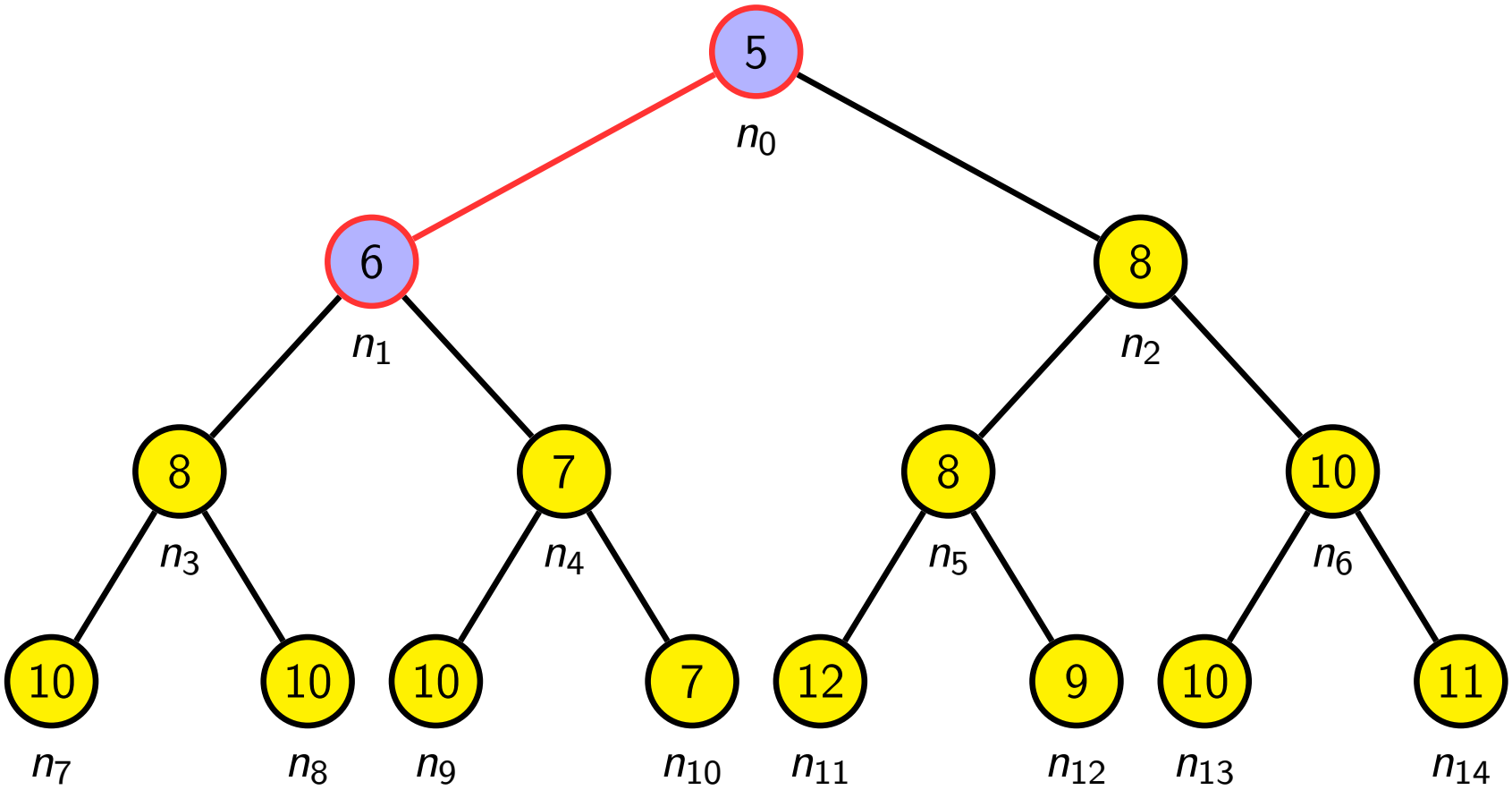
$\mathbf{f_{cur} = 0, f_{next} = \infty, ep_{cur} = <>, ep_{next} = <>}$



IDA* Suche mit Forced Deepening ($\lambda = 2$) und Elite Paths

1. Iteration:

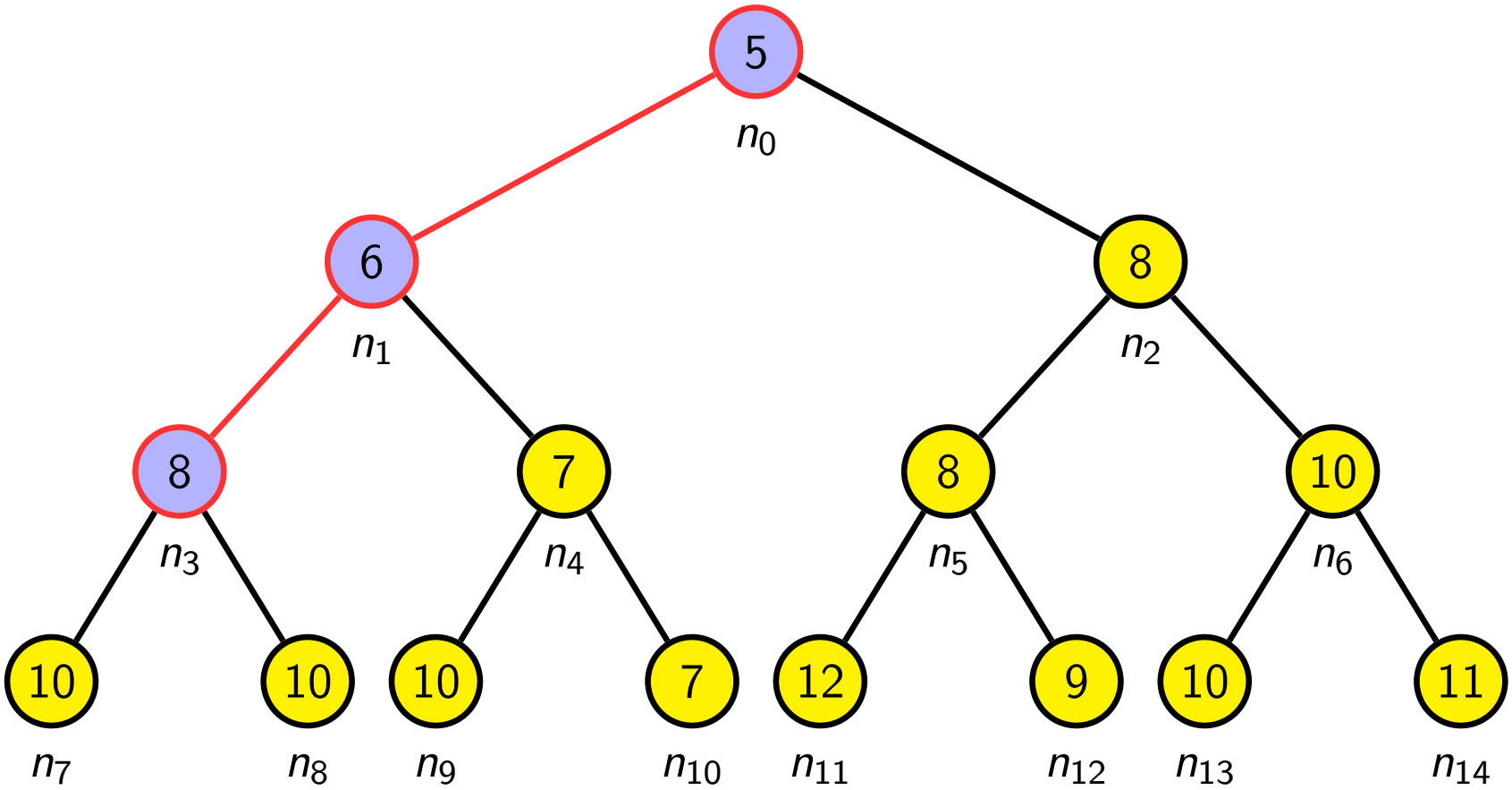
$\mathbf{f_{cur} = 0, f_{next} = \infty, ep_{cur} = <>, ep_{next} = <>}$



IDA* Suche mit Forced Deepening ($\lambda = 2$) und Elite Paths

1. Iteration:

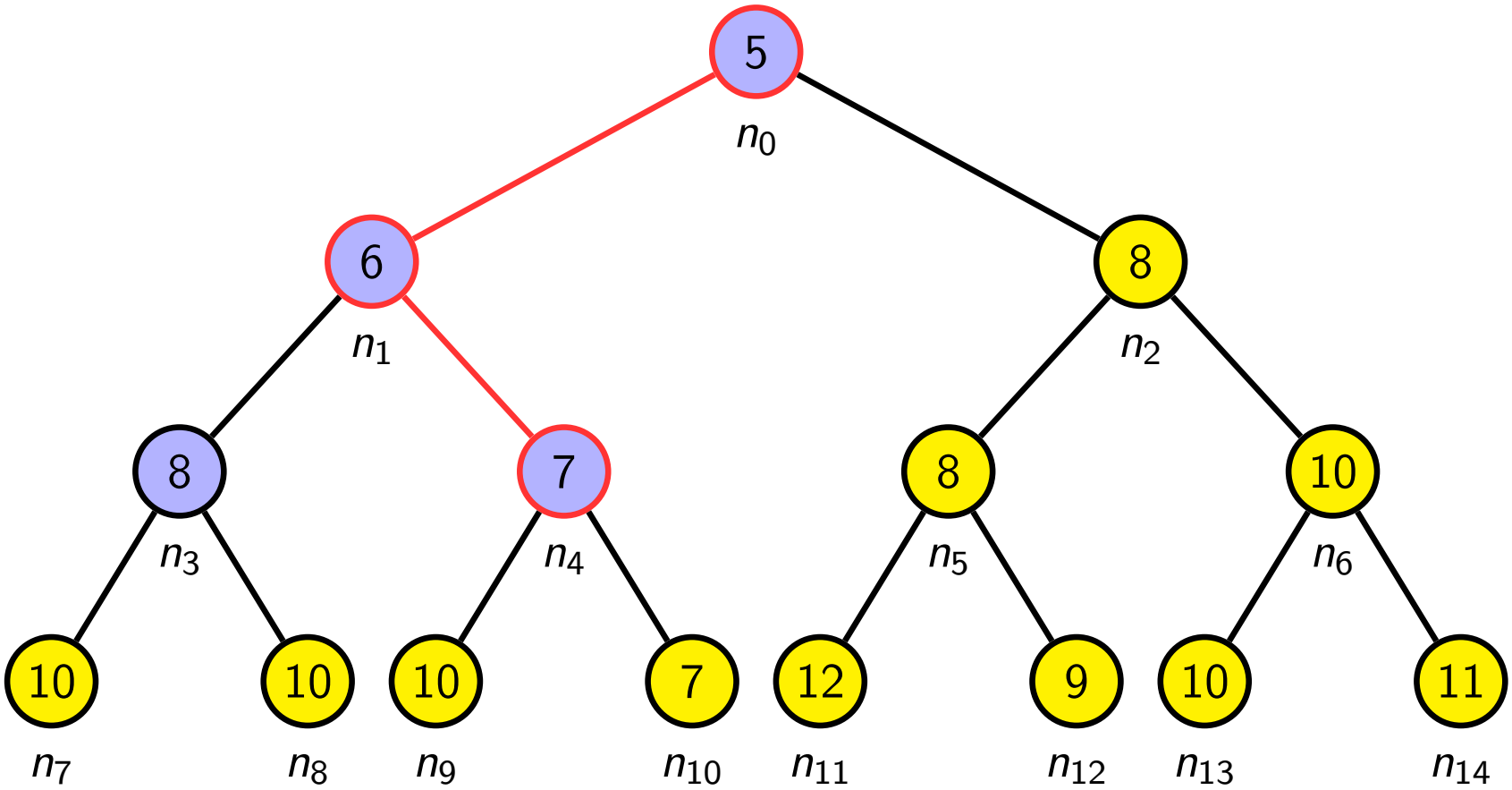
$\mathbf{f_{cur} = 0, f_{next} = 8, ep_{cur} =<>, ep_{next} =< n_0, n_1, n_3 >}$



IDA* Suche mit Forced Deepening ($\lambda = 2$) und Elite Paths

1. Iteration:

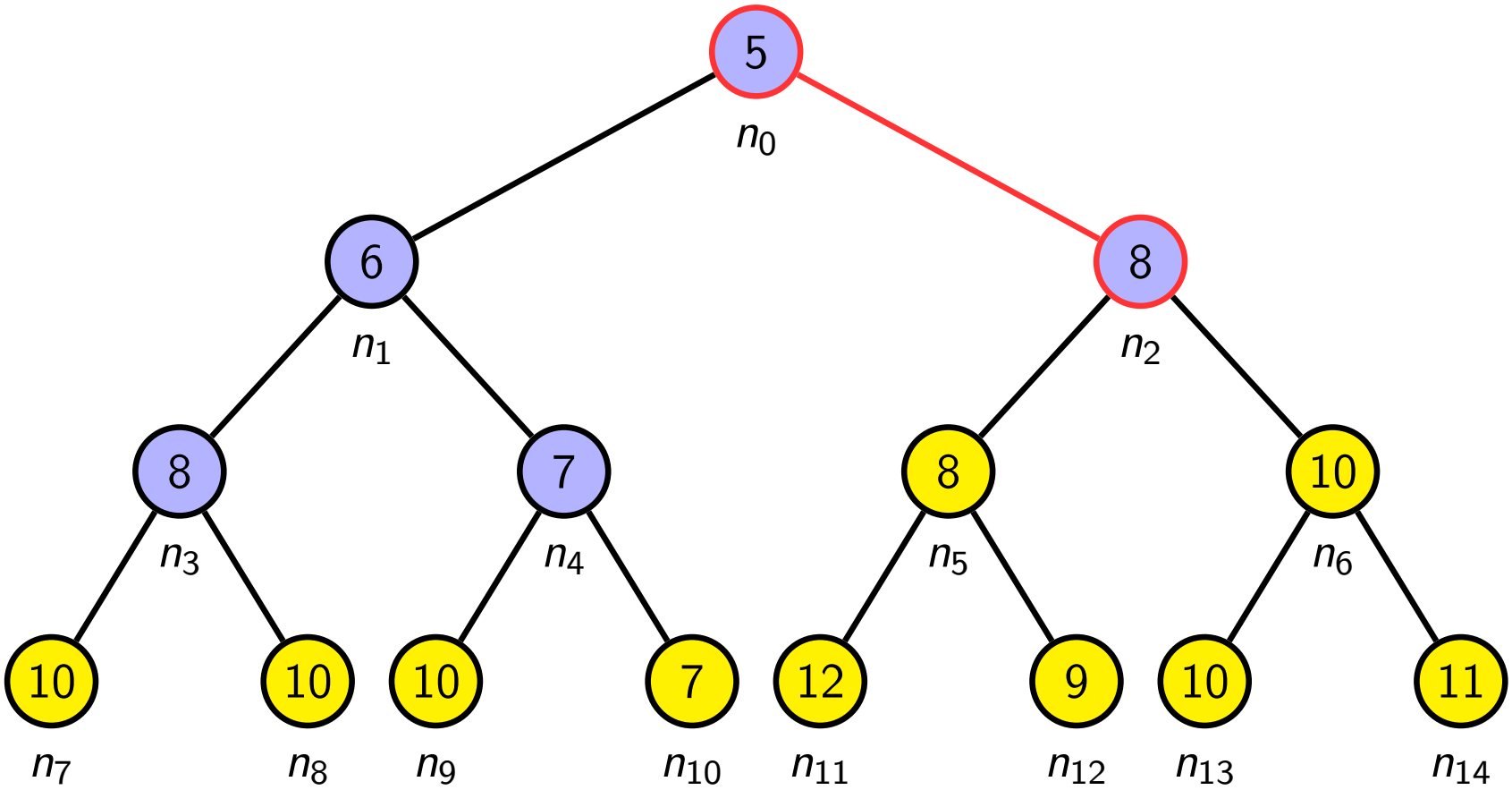
$\mathbf{f_{cur} = 0, f_{next} = 7, ep_{cur} = <>, ep_{next} = < n_0, n_1, n_4 >}$



IDA* Suche mit Forced Deepening ($\lambda = 2$) und Elite Paths

1. Iteration:

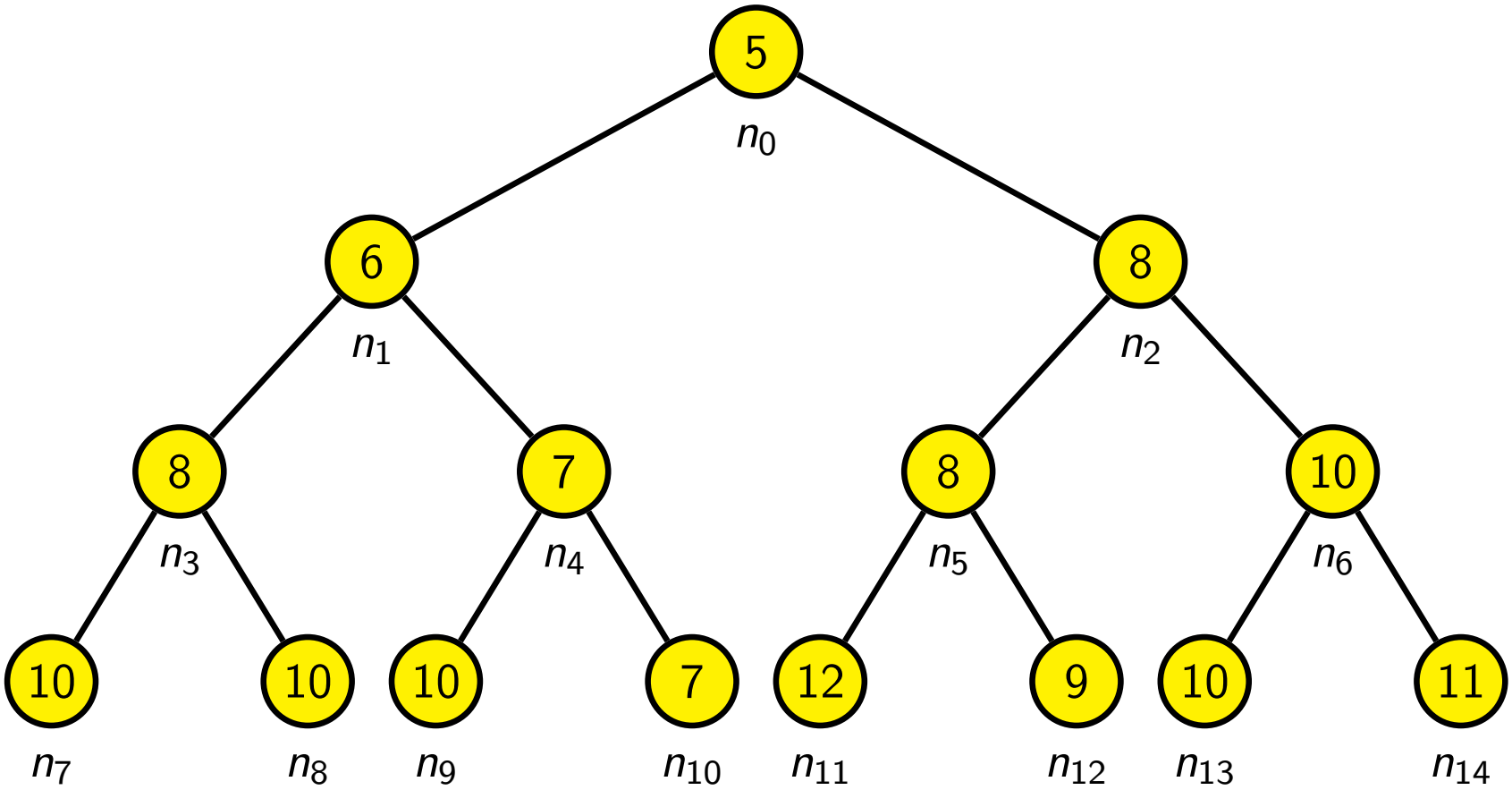
$\mathbf{f_{cur} = 0, f_{next} = 7, ep_{cur} =<>, ep_{next} =< n_0, n_1, n_4 >}$



IDA* Suche mit Forced Deepening ($\lambda = 2$) und Elite Paths

2. Iteration:

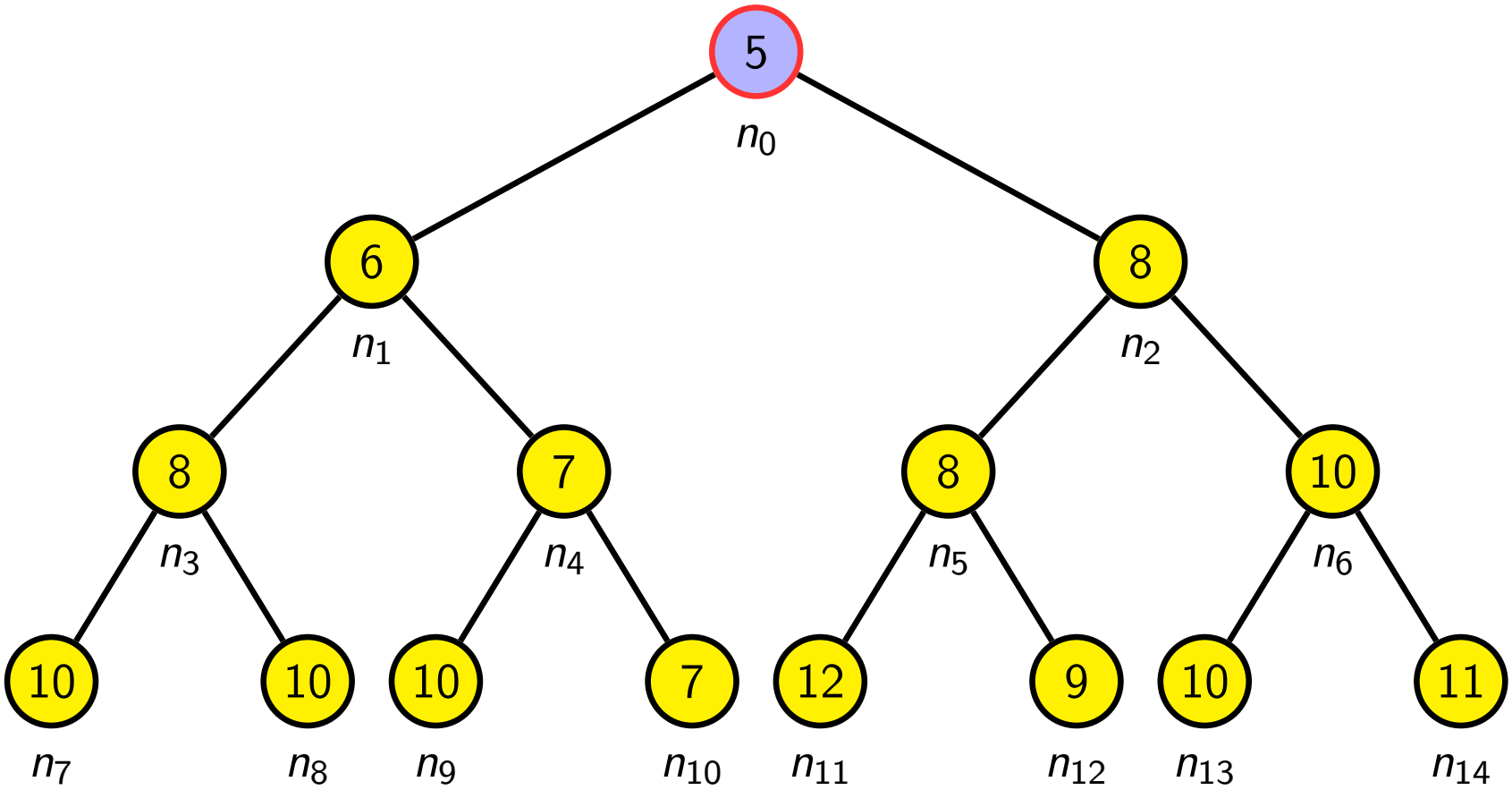
$\mathbf{f_{cur} = 7, f_{next} = \infty, ep_{cur} =< n_0, n_1, n_4 >, ep_{next} =<>}$



IDA* Suche mit Forced Deepening ($\lambda = 2$) und Elite Paths

2. Iteration:

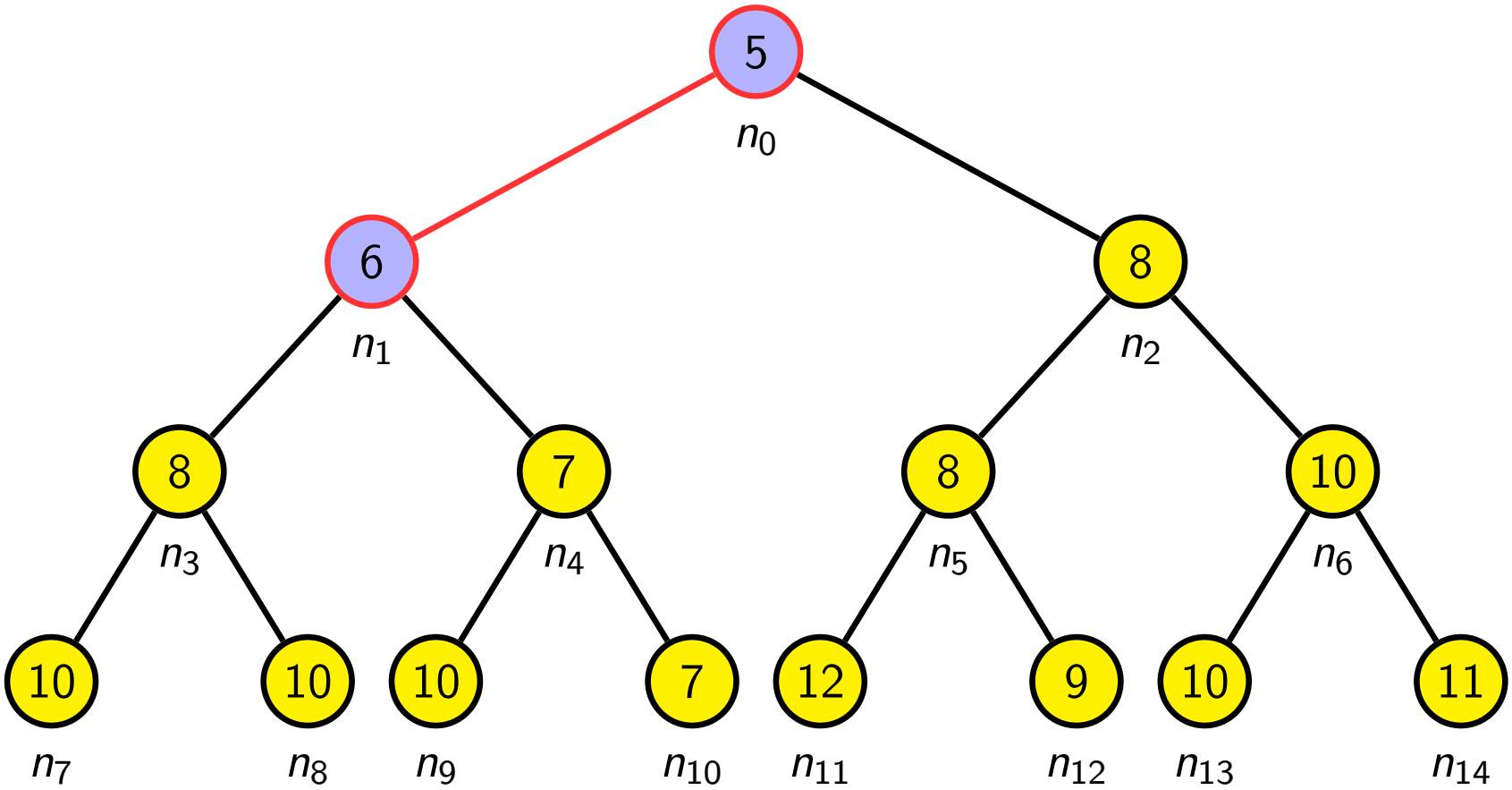
$\mathbf{f_{cur} = 7, f_{next} = \infty, ep_{cur} =< n_0, n_1, n_4 >, ep_{next} =<>}$



IDA* Suche mit Forced Deepening ($\lambda = 2$) und Elite Paths

2. Iteration:

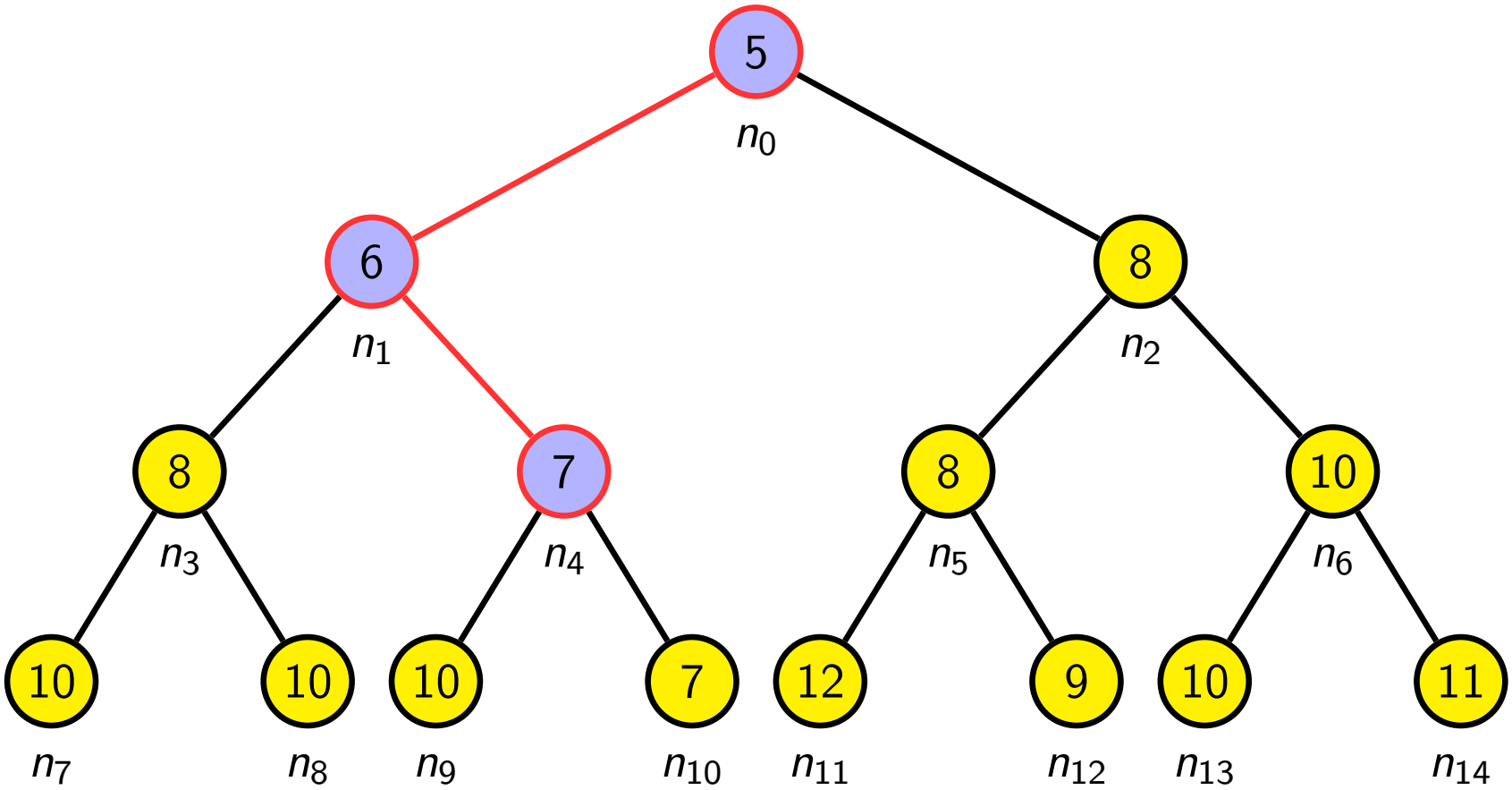
$\mathbf{f_{cur} = 7, f_{next} = \infty, ep_{cur} =< n_0, n_1, n_4 >, ep_{next} =<>}$



IDA* Suche mit Forced Deepening ($\lambda = 2$) und Elite Paths

2. Iteration:

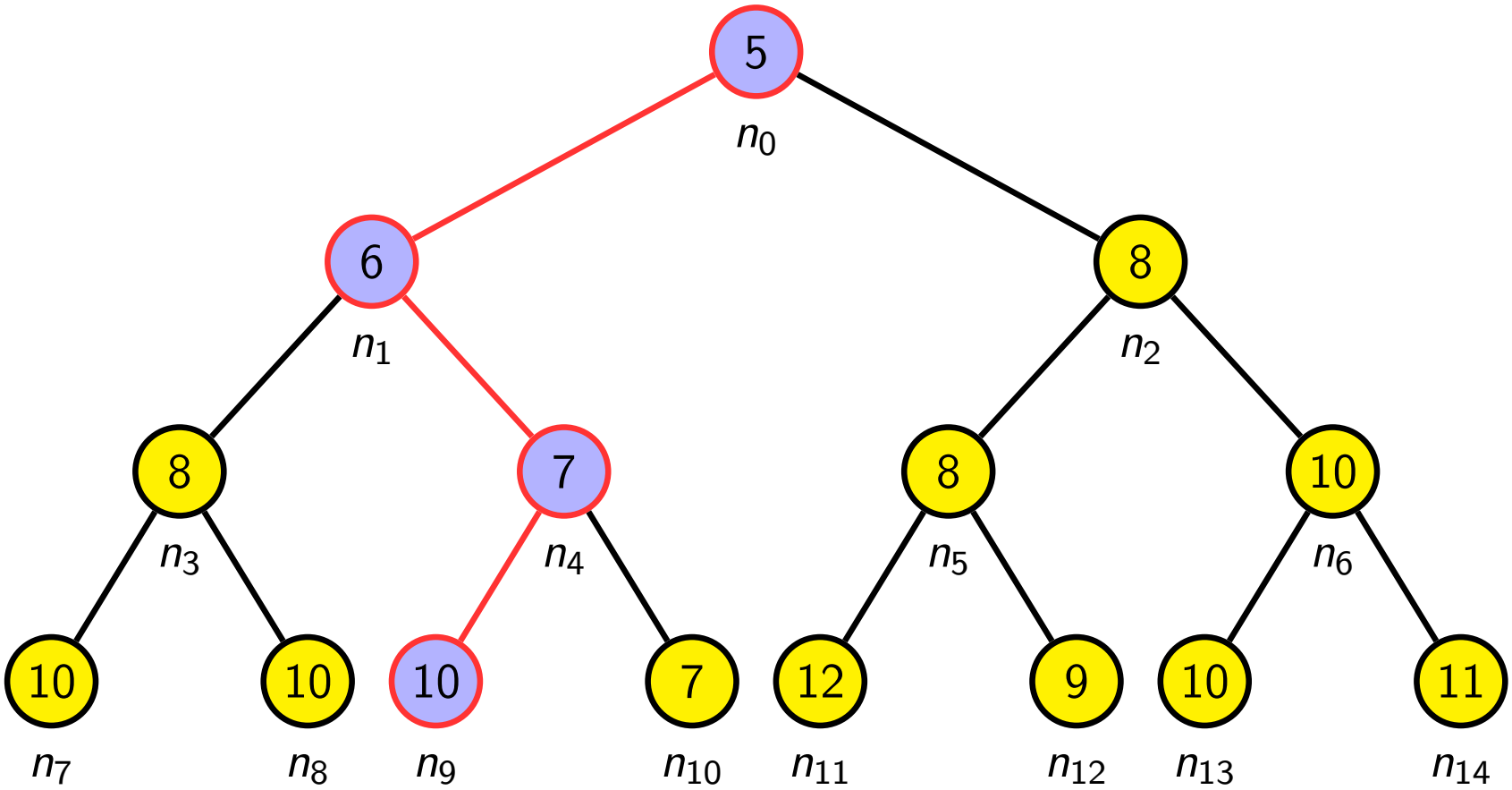
$\mathbf{f_{cur} = 7, f_{next} = \infty, ep_{cur} =< n_0, n_1, n_4 >, ep_{next} =<>}$



IDA* Suche mit Forced Deepening ($\lambda = 2$) und Elite Paths

2. Iteration:

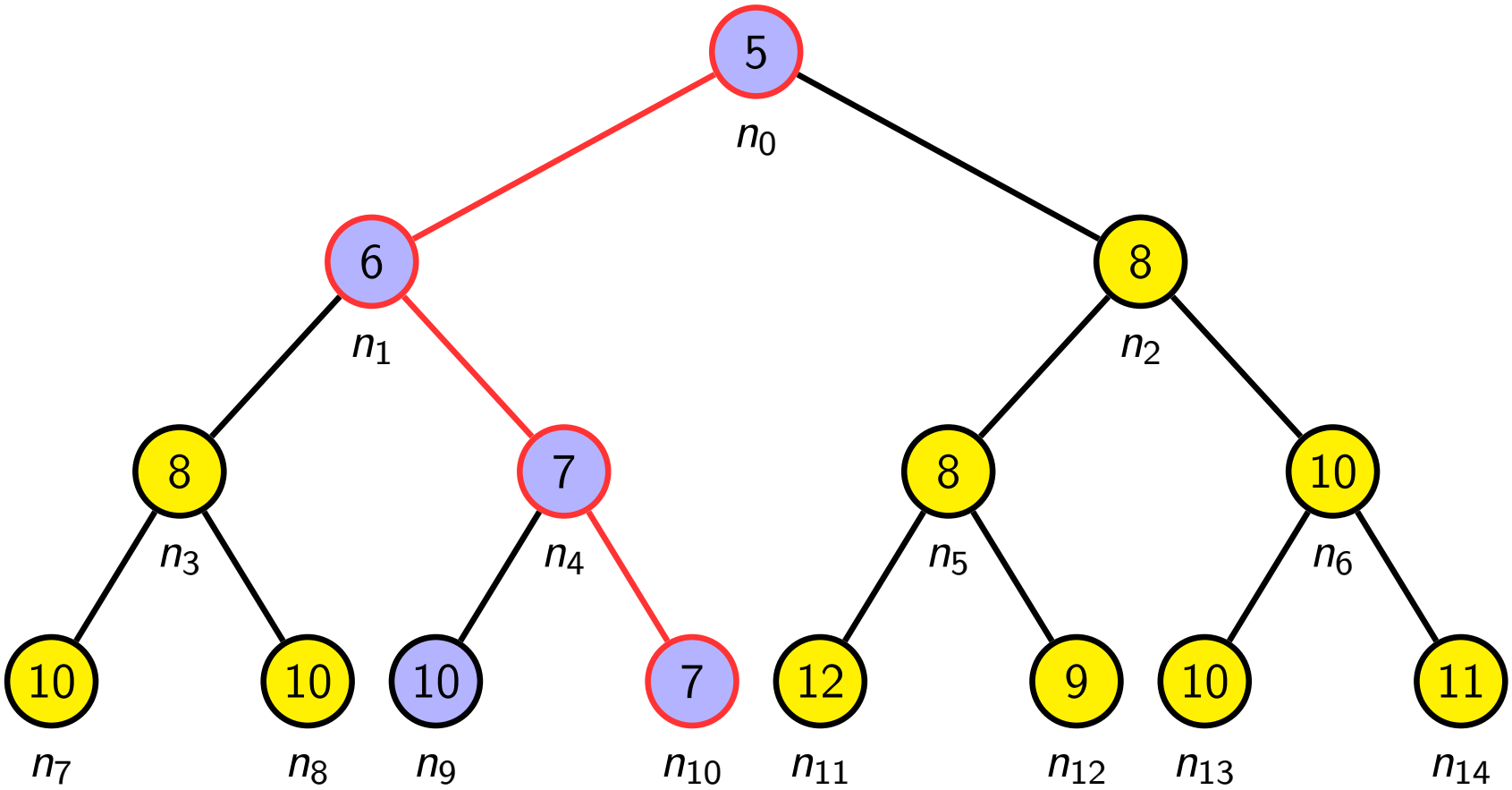
$\mathbf{f_{cur} = 7, f_{next} = 10, ep_{cur} =< n_0, n_1, n_4 >, ep_{next} =< n_0, n_1, n_4, n_9 >}$



IDA* Suche mit Forced Deepening ($\lambda = 2$) und Elite Paths

2. Iteration:

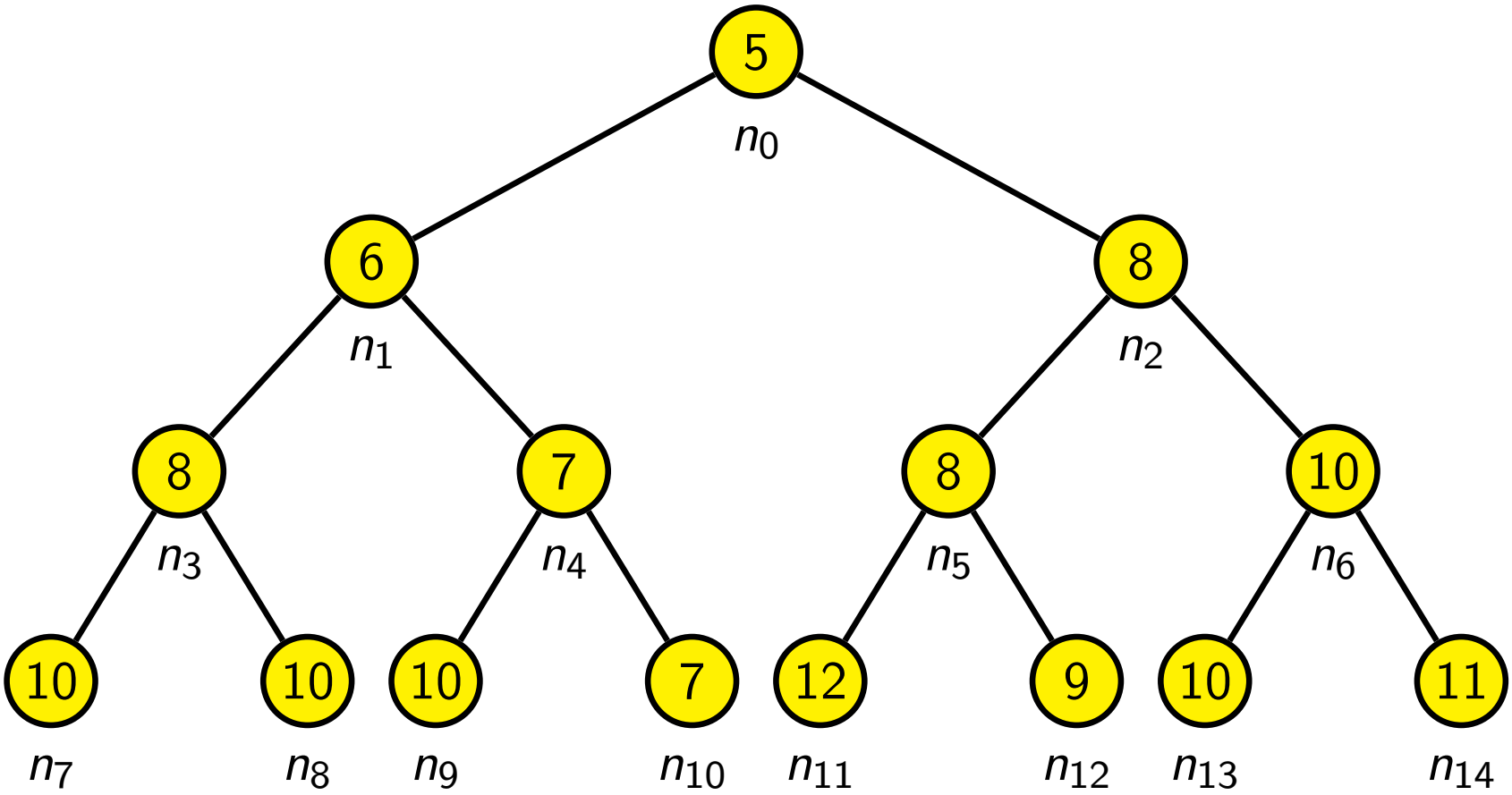
$\mathbf{f_{cur} = 7, f_{next} = 10, ep_{cur} =< n_0, n_1, n_4 >, ep_{next} =< n_0, n_1, n_4, n_9 >}$



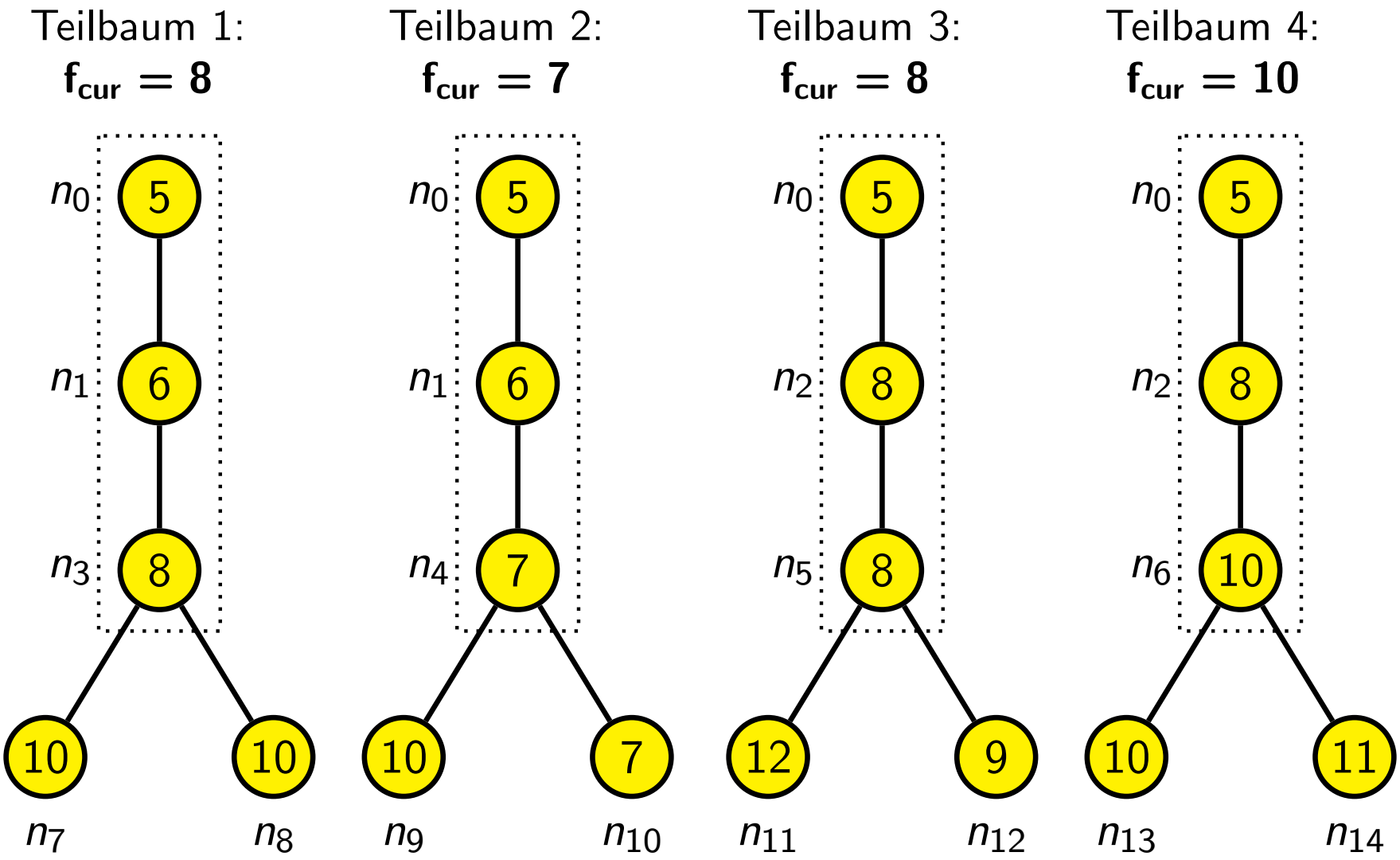
IDA* Suche mit Subtree Forest der Tiefe 2

IDA* Suche mit Subtree Forest der Tiefe 2

ganzer Suchbaum



IDA* Suche mit Subtree Forest der Tiefe 2



Verbesserungen

Multithreading

Um die Performance der Suche zu verbessern habe ich einige weitere Verbesserungen in meine Implementierung eingebunden. Die Wichtigsten stelle ich hier kurz vor:

- **Multithreading**: Wenn Subtree Forests angewendet werden, können mehrere Teilbäume parallel untersucht werden. So kann die Rate der expandierten Knoten pro Sekunde massiv erhöht werden.

Disjoint Pattern Database und Team Cache

- **Disjoint Pattern Database**: Heuristikwerte können für alle Teams vorberechnet und im Speicher gehalten werden.
- **Team Cache**: Heuristikwerte können entlang eines Pfades im Suchbaum für alle Teams gespeichert werden. Wenn die Heuristik für einen neuen Knoten berechnet werden muss, ändern sich nur die Heuristikwerte für die beiden Teams, die das hinzugefügte Spiel bestreiten.

Evaluation

Einfluss von Forced Deepening, Elite Paths und Subtree Forests

FD	EP	SF	SUPER4	GALAXY4	SUPER6	GALAXY6	SUPER8	GALAXY8
×	×	×	100	100	100	100	-	100
×	×	✓	**45.31**	**35.91**	35.41	96.95	-	99.70
×	✓	×	83.20	83.43	99.99	100.58	-	101.07
×	✓	✓	54.30	37.02	36.01	97.10	-	99.70
✓	×	×	278.91	141.44	0.57	36.83	100	17.23
✓	×	✓	91.80	63.54	**0.16**	12.23	97.18	10.09
✓	✓	×	103.52	72.10	0.20	**10.73**	**92.25**	11.95
✓	✓	✓	81.64	57.46	0.18	11.65	94.37	**9.77**

relative Anzahl expandierter Knoten für verschiedene Probleminstanzen, alle Zahlen erhalten ohne Einsatz von Multithreading

Einfluss von Forced Deepening, Elite Paths und Subtree Forests

FD	EP	SF	SUPER4	GALAXY4	SUPER6	GALAXY6	SUPER8	GALAXY8
×	×	×	100	100	100	100	-	100
×	×	✓	**45.31**	**35.91**	35.41	96.95	-	99.70
×	✓	×	83.20	83.43	99.99	100.58	-	101.07
×	✓	✓	54.30	37.02	36.01	97.10	-	99.70
✓	×	×	278.91	141.44	0.57	36.83	100	17.23
✓	×	✓	91.80	63.54	**0.16**	12.23	97.18	10.09
✓	✓	×	103.52	72.10	0.20	**10.73**	**92.25**	11.95
✓	✓	✓	81.64	57.46	0.18	11.65	94.37	**9.77**

Forced Deepening mit $\lambda =$ Anzahl Teams:

- stark positiver Einfluss bei SUPER6
- eher schädlich bei Instanzen mit 4 Teams

Einfluss von Forced Deepening, Elite Paths und Subtree Forests

FD	EP	SF	SUPER4	GALAXY4	SUPER6	GALAXY6	SUPER8	GALAXY8
×	×	×	100	100	100	100	-	100
×	×	✓	**45.31**	**35.91**	35.41	96.95	-	99.70
×	✓	×	83.20	83.43	99.99	100.58	-	101.07
×	✓	✓	54.30	37.02	36.01	97.10	-	99.70
✓	×	×	278.91	141.44	0.57	36.83	100	17.23
✓	×	✓	91.80	63.54	**0.16**	12.23	97.18	10.09
✓	✓	×	103.52	72.10	0.20	**10.73**	**92.25**	11.95
✓	✓	✓	81.64	57.46	0.18	11.65	94.37	**9.77**

Elite Paths:

- generell weniger Einfluss als Forced Deepening
- häufig auch negativer Einfluss

Einfluss von Forced Deepening, Elite Paths und Subtree Forests

FD	EP	SF	SUPER4	GALAXY4	SUPER6	GALAXY6	SUPER8	GALAXY8
×	×	×	100	100	100	100	-	100
×	×	✓	**45.31**	**35.91**	35.41	96.95	-	99.70
×	✓	×	83.20	83.43	99.99	100.58	-	101.07
×	✓	✓	54.30	37.02	36.01	97.10	-	99.70
✓	×	×	278.91	141.44	0.57	36.83	100	17.23
✓	×	✓	91.80	63.54	**0.16**	12.23	97.18	10.09
✓	✓	×	103.52	72.10	0.20	**10.73**	**92.25**	11.95
✓	✓	✓	81.64	57.46	0.18	11.65	94.37	**9.77**

Subtree Forest mit Tiefe $\frac{\text{Anzahl Teams}}{2}$:

- fast immer positiver Einfluss
- Ausnahme: wenn FD und EP aktiviert

Einfluss der anderen Verbesserungen

- **Multithreading**:
  - starker Anstieg der expandierten Knoten pro Sekunde beobachtbar, wenn Anzahl Threads erhöht wird
  - nur solange Anzahl Threads $\leq$ Anzahl Rechenkerne
- **Disjoint Pattern Database** und **Team Cache**:
  - starker Gewinn an Zeit-Performance für beide
  - Effekt ist um ein Vielfaches stärker für DPD
  - ohne DPD: nur Instanzen mit 4 Team können in vernünftiger Zeit gelöst werden

Fragen?