自然言語処理プログラミング勉強会 13
ビーム探索と A* 探索

Graham Neubig
奈良先端科学技術大学院大学 (NAIST)

予測問題

- 観測された情報 X に基づいて Y を予測

$$\underset{Y}{\operatorname{argmax}}\, P\left(Y \middle| X\right)$$

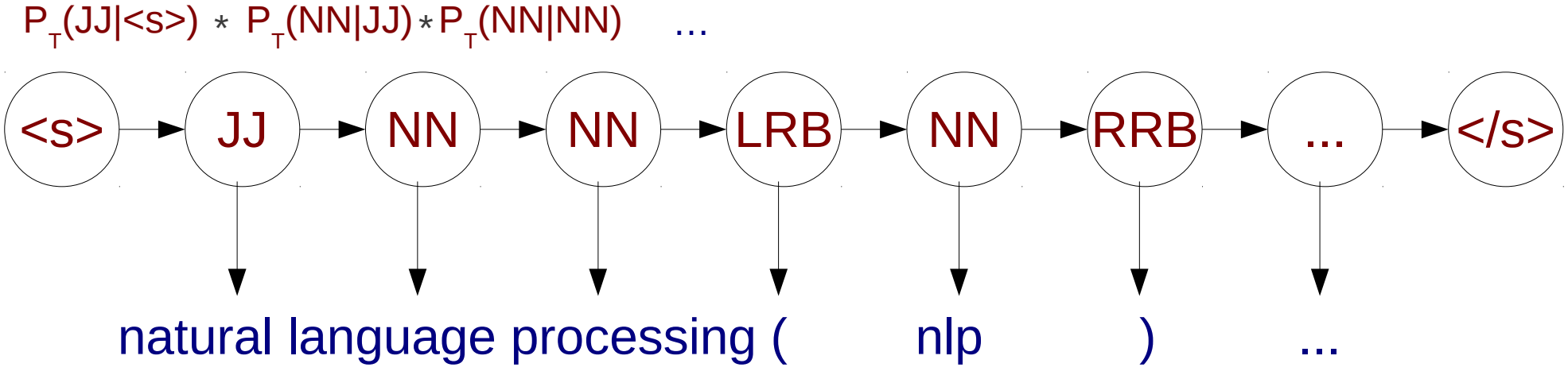
- 品詞推定、単語分割、構文解析などで利用
- argmax を解くのが探索
- 今まで主にビタビアルゴリズムを利用

品詞推定のための (HMM)

- 品詞→品詞の遷移確率
  - 2-gram モデルとほぼ一緒
- 品詞→単語の生成確率

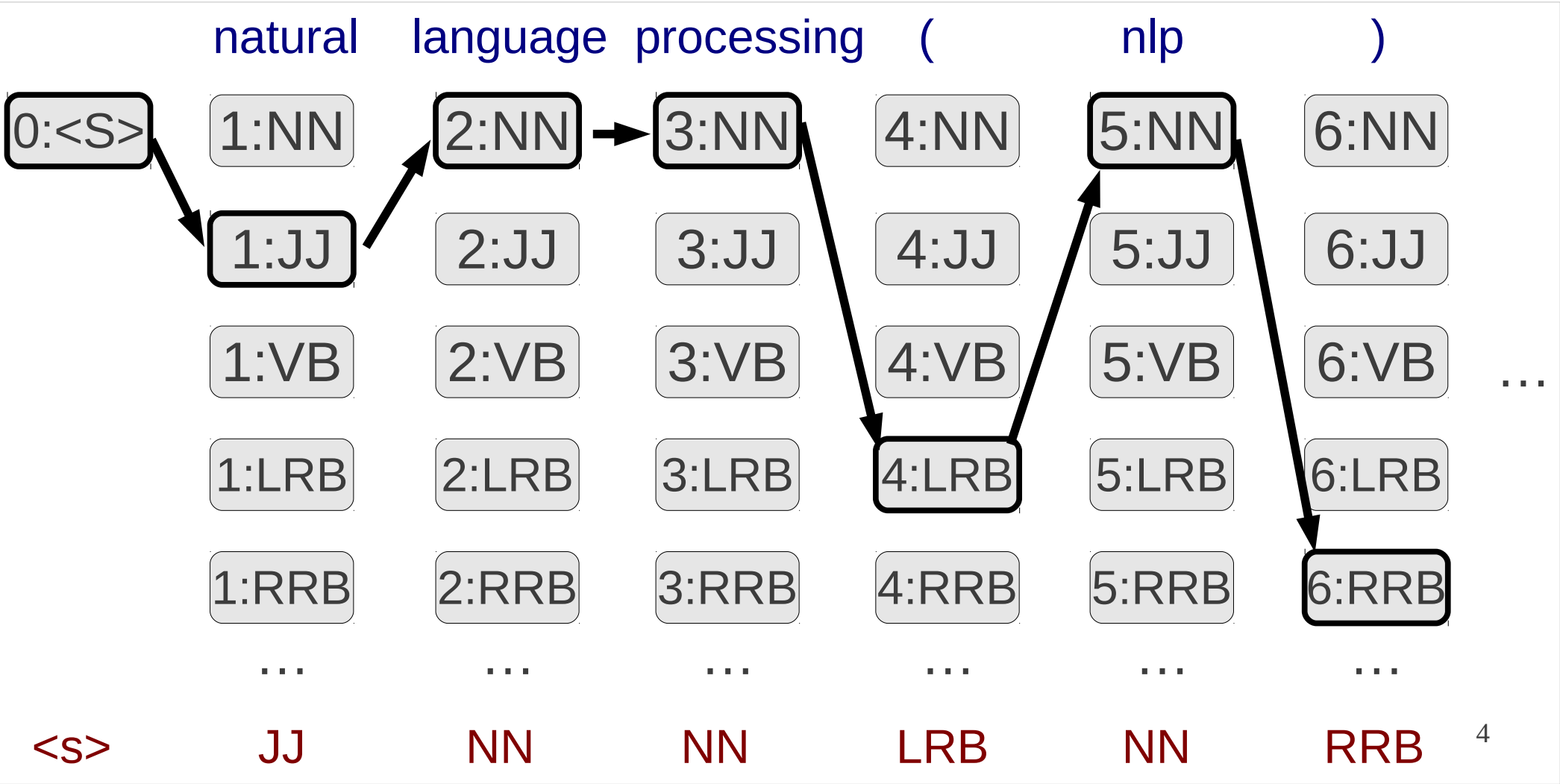
$$P(Y) \approx \prod_{i=1}^{I+1} P_T(y_i | y_{i-1})$$

$$P(X|Y) \approx \prod_{1}^{I} P_E(x_i | y_i)$$



HMM 品詞推定のグラフ

- 各パスは品詞列を表す

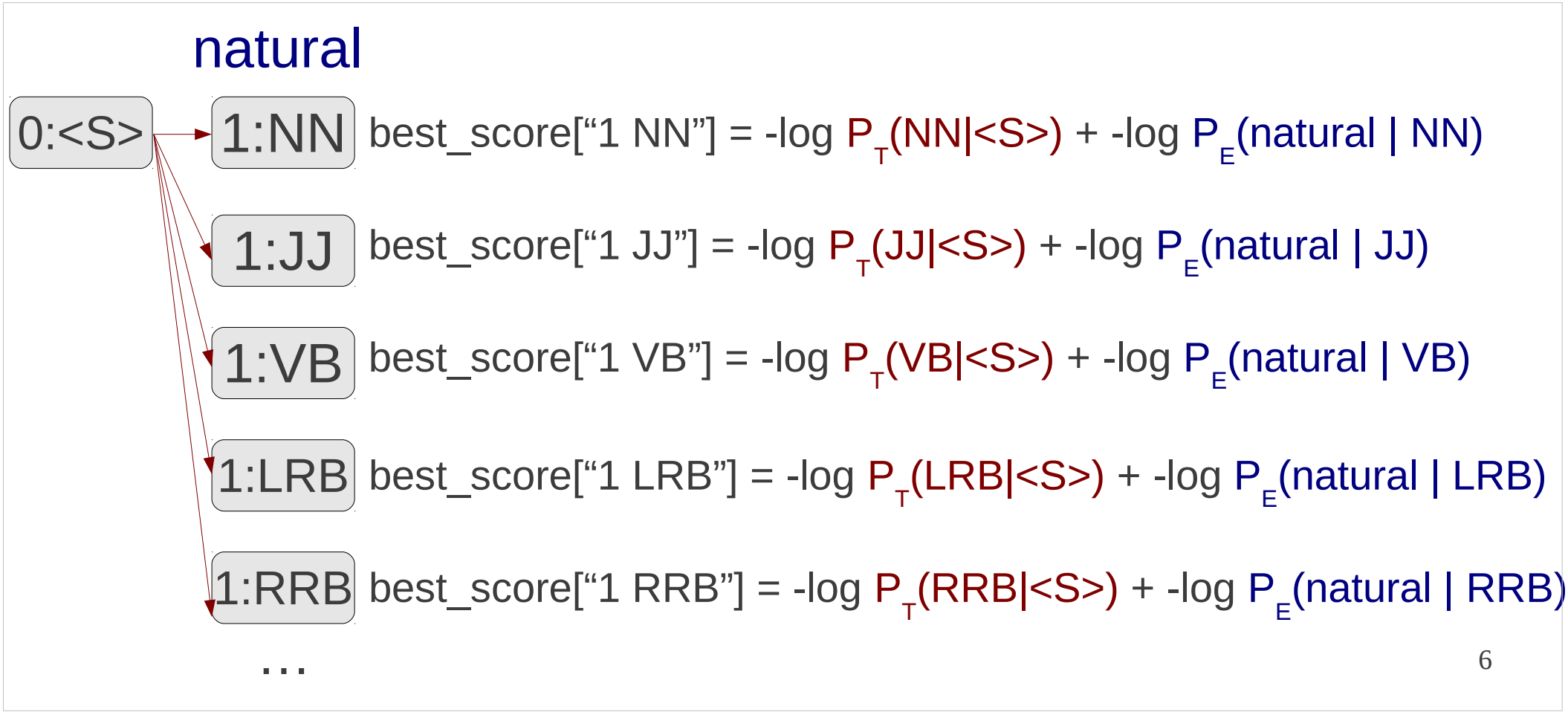


復習：ビタビアルゴリズムのステップ

- 前向きステップ：各ノードへたどる確率の計算
  - 負の対数尤度がもっとも低くなるパス
- 後ろ向きステップ：パスの復元
  - 単語分割とほとんど同じ

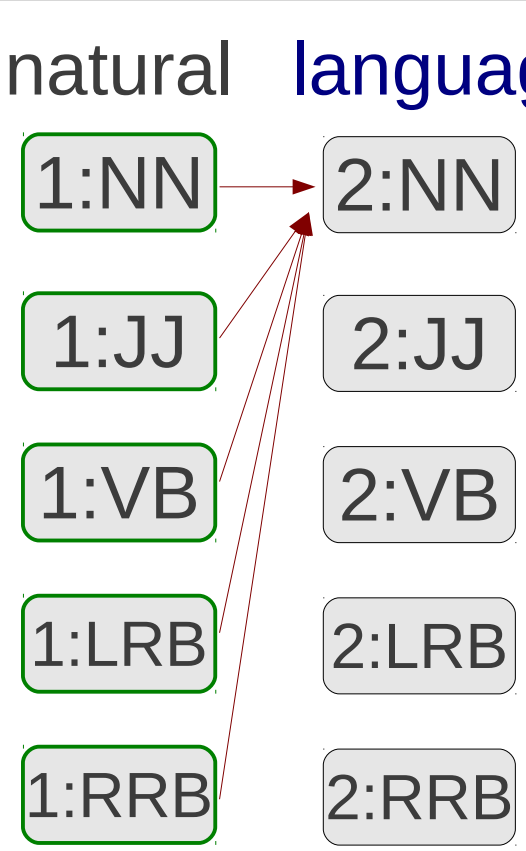
前向きステップ：文頭

- 文頭記号 <S> から 1 単語目への遷移と 1 単語目の生成の確率



前向きステップ：中間

- 前の品詞を全部比べて、これまでのパス、遷移、生成を全て考慮した最短パスを利用

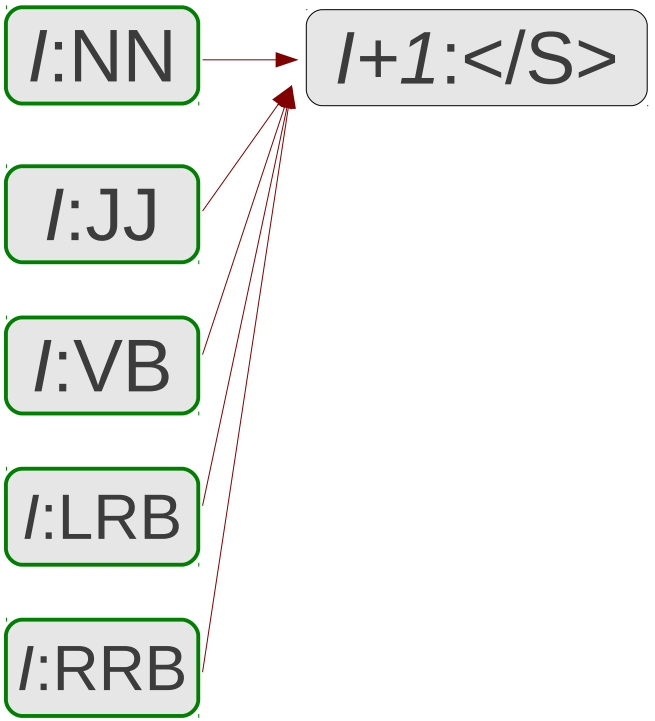


best_score[“2 NN”] = min(
best_score[“1 NN”] + -log P_T(NN|NN) + -log P_E(language | NN),
best_score[“1 JJ”] + -log P_T(NN|JJ) + -log P_E(language | NN),
best_score[“1 VB”] + -log P_T(NN|VB) + -log P_E(language | NN),
best_score[“1 LRB”] + -log P_T(NN|LRB) + -log P_E(language | NN),
best_score[“1 RRB”] + -log P_T(NN|RRB) + -log P_E(language | NN),
...
)
best_score[“2 JJ”] = min(
best_score[“1 NN”] + -log P_T(JJ|NN) + -log P_E(language | JJ),
best_score[“1 JJ”] + -log P_T(JJ|JJ) + -log P_E(language | JJ),
best_score[“1 VB”] + -log P_T(JJ|VB) + -log P_E(language | JJ),
...

前向きステップ：文末

- 文末記号への遷移を考慮して終わり

science



best_score[“*I+1* </S>”] = min(
  best_score[“*I* NN”] + -log P_T(</S>|NN),
  best_score[“*I* JJ”] + -log P_T(</S>|JJ),
  best_score[“*I* VB”] + -log P_T(</S>|VB),
  best_score[“*I* LRB”] + -log P_T(</S>|LRB),
  best_score[“*I* NN”] + -log P_T(</S>|RRB),
  …
)

ビタビアルゴリズムと計算量

- ビタビアルゴリズムのかかる時間に影響する要因：
  - 問題の種類：品詞推定？ 単語分割？ 構文解析？
  - 文の長さ：文が長い＝時間も長い
  - タグの種類：タグが多い＝時間も長い
    などなど
- 品詞推定の計算量は？
  - T＝ タグの数
  - N＝ 文の長さ

単純なビタビはスケールしない！

- タグ付け
  - 固有表現抽出 :
    T = 固有表現の種類数 (100~10,000)
  - スーパータギング :
    T = 文法ルールの数 (100~1,000)
- より難しい探索問題
  - 構文解析 : $T * N^3$
  - 音声認識 : フレーム数 (1,000?)*WFST 状態 (1,000,000?)
  - 機械翻訳 : NP 完全

２つの解決策

- ビーム探索 :
  - 低確率の仮説を枝刈り
  - + 簡単、探索時間は推測可能
  - - 近似解なので、精度の低下あり
- A* 探索 :
  - 深さ優先探索
    ヒューリスティック関数でスピード向上
  - + ビタビや深さ優先より速い、厳密なスコア最大化
  - - ヒューリスティック関数の作成が必要、探索時間にゆれが存在

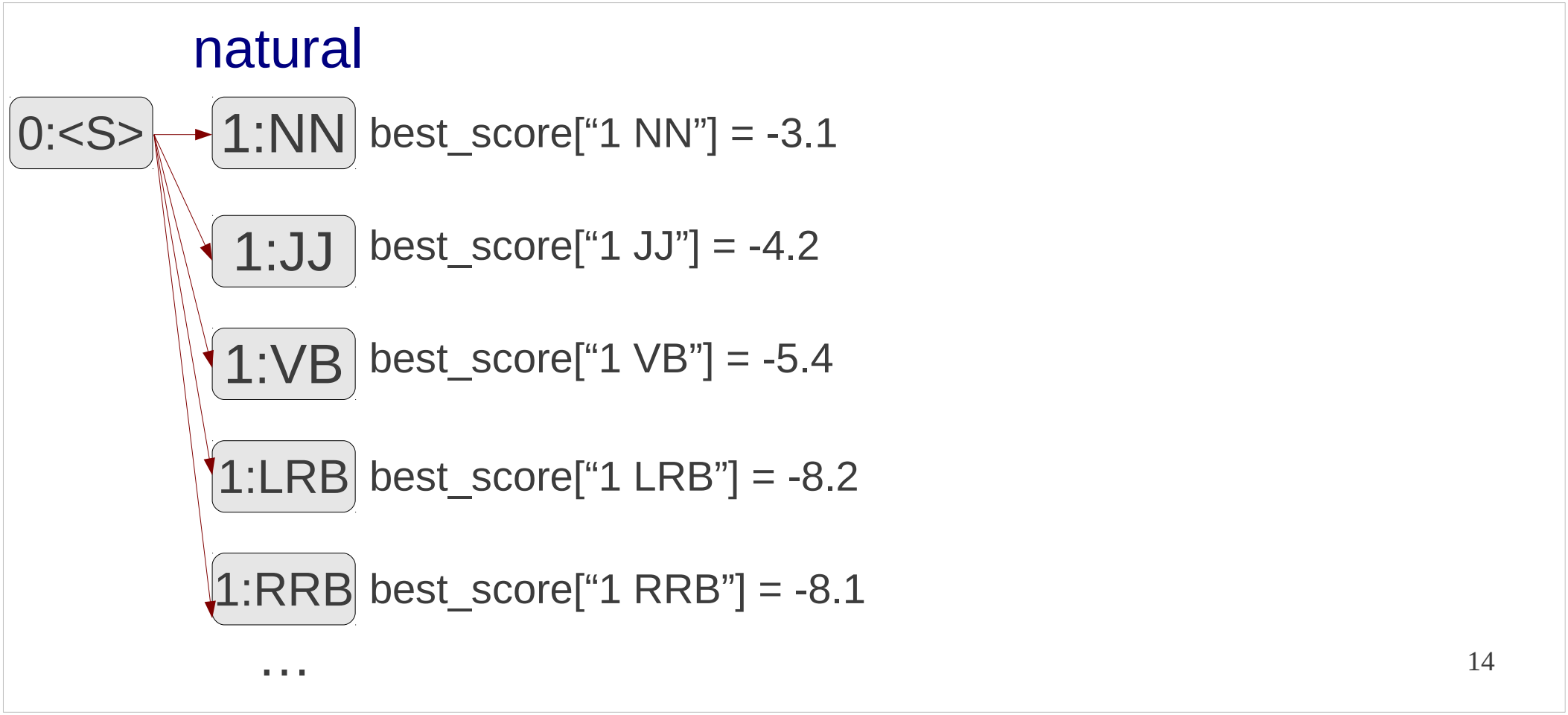
ビーム探索

ビーム探索

- B 個の仮説からなるビームを利用
- ビタビアルゴリズム行うが、各段階において B 個の仮説しか展開しない
- 比較対象（段階）はタスクによる：
  - タグ付け: タグ済みの単語数が同一の仮説を比較
  - 機械翻訳: 翻訳済みの単語数が同一の仮説を比較
  - 音声認識: 処理済みのフレーム数が同一の仮説を比較

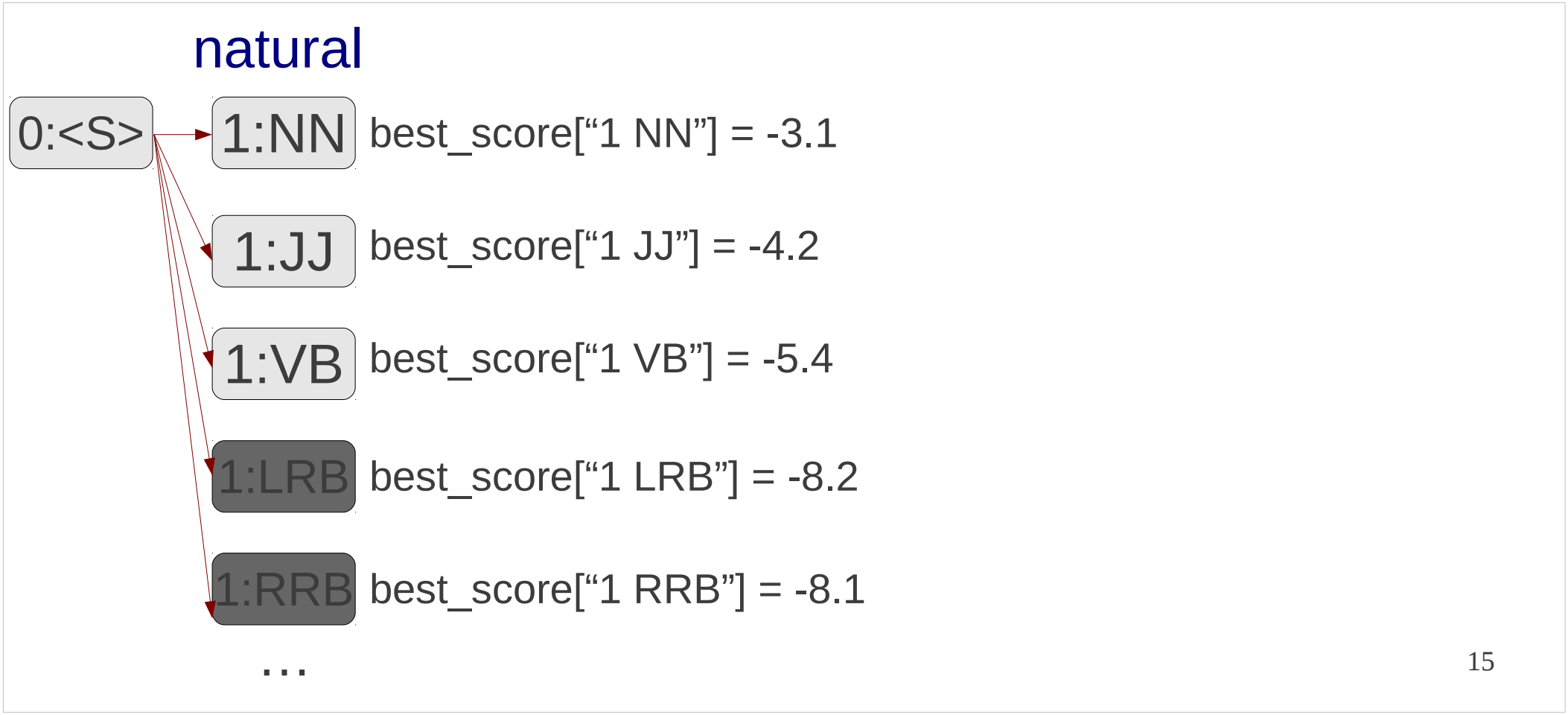
スコア計算 (1 単語目)

- 最初の単語のスコア最大のパスを計算



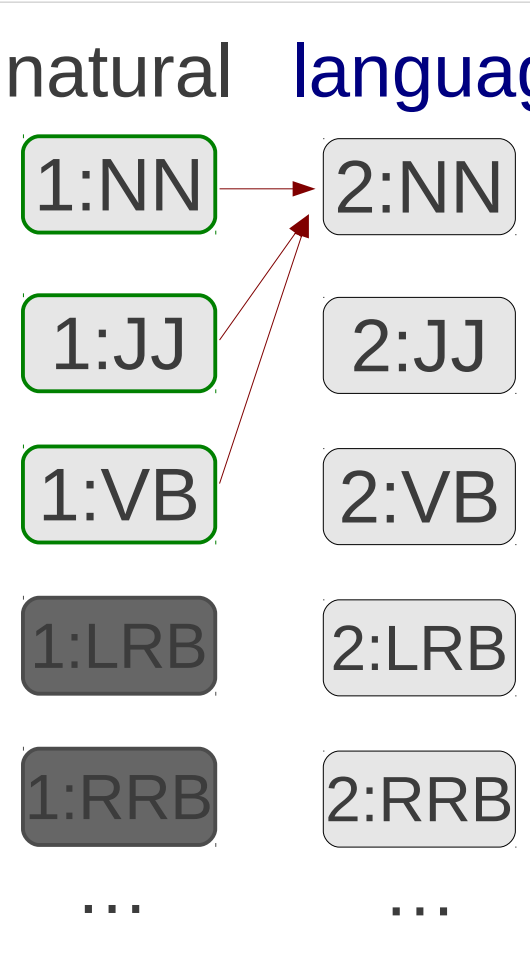
スコア最大の B 個の仮説を残す (w_1)

- 低い仮説のスコアを削除
- 例えば B=3 の場合：



スコア計算 (w_2)

- スコアは計算するが、削除された仮説は考慮しない



best_score[“2 NN”] = min(
best_score[“1 NN”] + -log P_T(NN|NN) + -log P_E(language | NN),
best_score[“1 JJ”] + -log P_T(NN|JJ) + -log P_E(language | NN),
best_score[“1 VB”] + -log P_T(NN|VB) + -log P_E(language | NN),
~~best_score[“1 LRB”] + -log P_T(NN|LRB) + -log P_E(language | NN),~~
~~best_score[“1 RRB”] + -log P_T(NN|RRB) + -log P_E(language | NN),~~
...
)
best_score[“2 JJ”] = min(
best_score[“1 NN”] + -log P_T(JJ|NN) + -log P_E(language | JJ),
best_score[“1 JJ”] + -log P_T(JJ|JJ) + -log P_E(language | JJ),
best_score[“1 VB”] + -log P_T(JJ|VB) + -log P_E(language | JJ),
...

ビーム探索はビタビより速い！

- ある仮説を考慮しなくても良い分、速い
- 計算量はどれぐらい？
  - T = タグ数
  - N = 文の長さ
  - B = ビーム幅

実装：前向きステップ

best_score[“0 <s>”] = 0　# <s> で開始
best_edge[“0 <s>”] = NULL
active_tags[0] = [“<s>”]
for *i* **in** 0 … *l*-1:
　make map *my_best*
　for each *prev* **in** keys of *active_tags*[*i*]
　　for each *next* **in** keys of *possible_tags*
　　if best_score[“*i prev*”] **and** transition[“prev next”] **exist**
　　score = best_score[“i prev”] +
　　-log P_T(next|prev) + -log P_E(word[i]|next)
　　if *best_score*[“*i+1 next*”] **is** *new* **or** > *score*
　　best_score[“*i+1 next*”] = *score*
　　best_edge[“*i+1 next*”] = “*i prev*”
　　my_best[*next*] = *score*
　active_tags[*i+1*] = best *B* elements of *my_best*
</s> で同等の処理を行う

A* 探索

深さ優先探索

- 必ずスコア最大の仮説を展開
- 仮説をヒープ（優先度つきキュー）で管理
  - ヒープ: O(1) 時間で仮説が追加可能、 O(log n) でスコア最大の仮説が探索＋削除可能なデータ構造
  - まず開始状態をヒープに追加
  - 探索終了までスコア最大の仮説を展開
- 1 段階ずつ仮説を展開する幅優先探索（ビタビ、ビーム）と比較

深さ優先探索

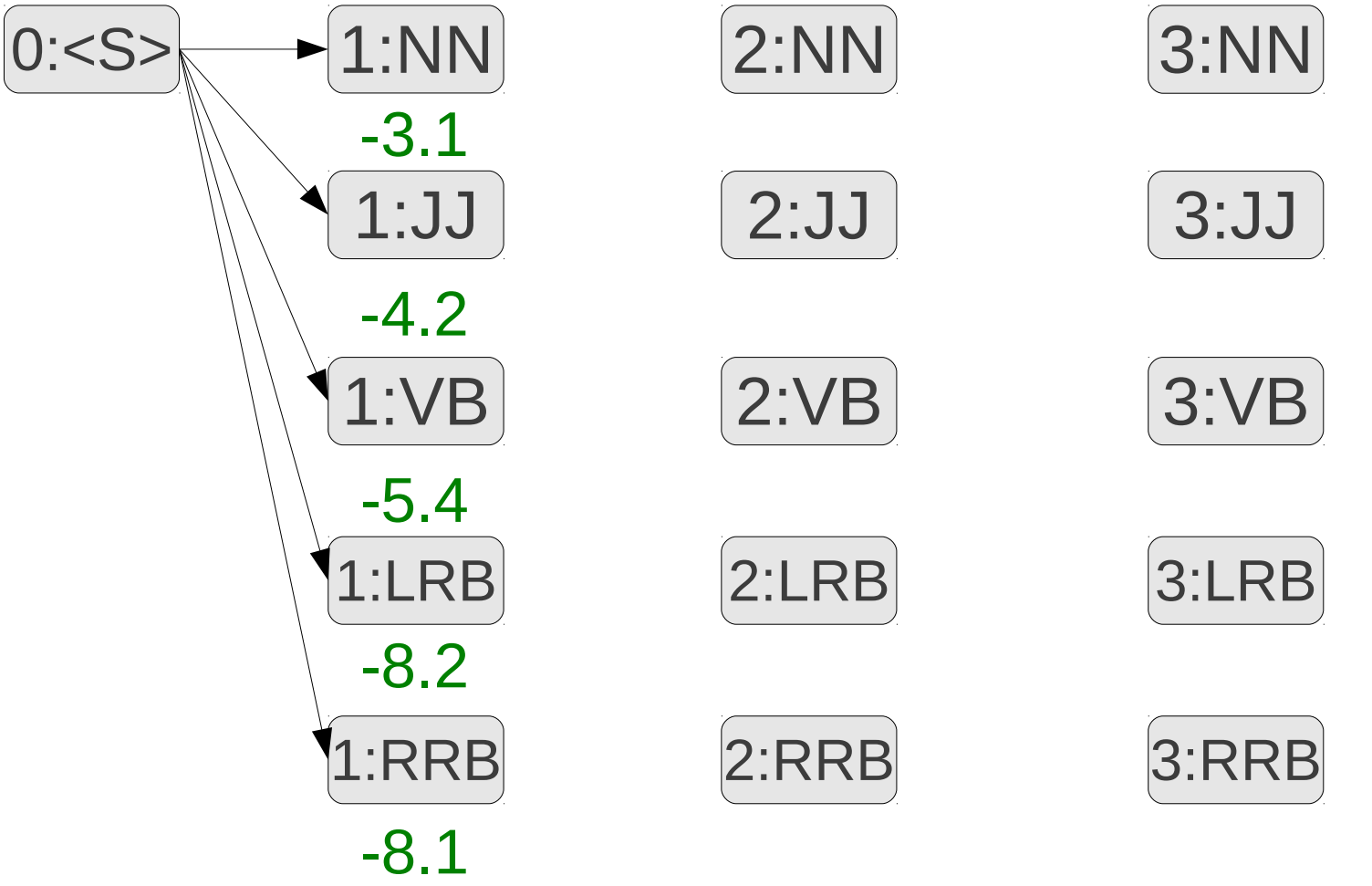
- 初期状態

ヒープ

0:<S>　0

深さ優先探索

- 0:<S> を処理

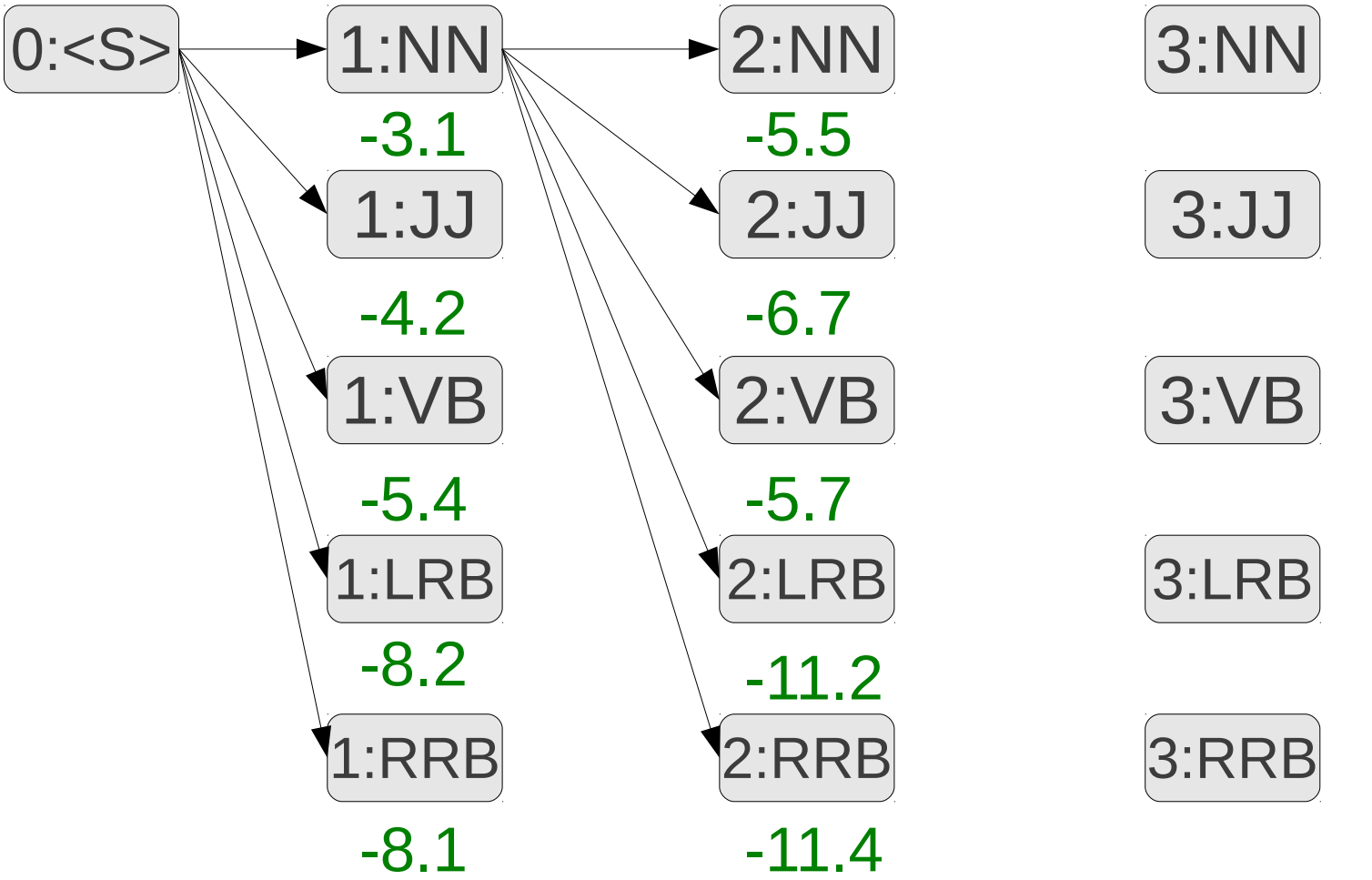


ヒープ

1:NN -3.1
1:JJ -4.2
1:VB -5.4
1:RRB -8.1
1:LRB -8.2

深さ優先探索

- 1:NN を処理

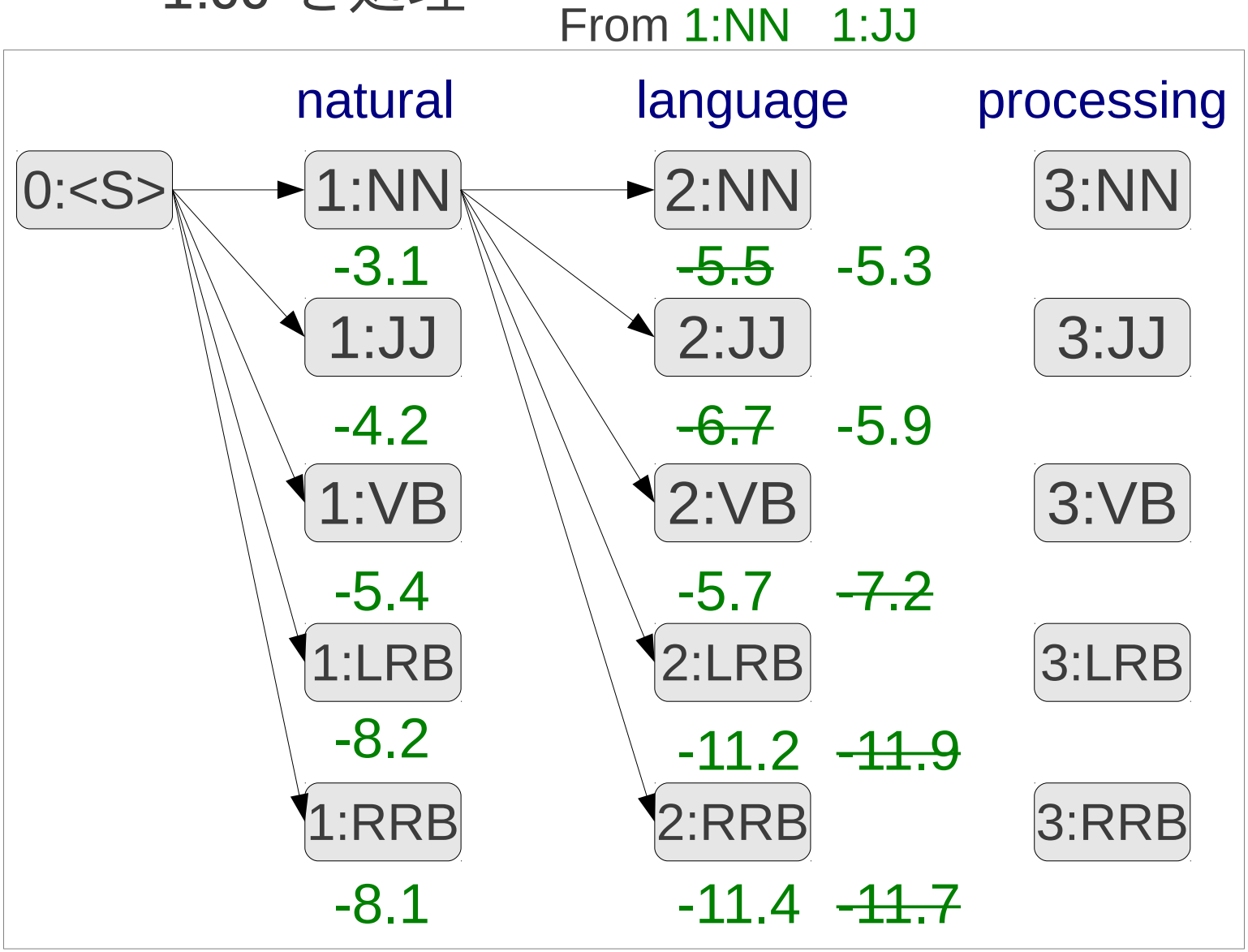


ヒープ

1:JJ -4.2
1:VB -5.4
2:NN -5.5
2:VB -5.7
2:JJ -6.7
1:RRB -8.1
1:LRB -8.2
2:LRB -11.2
2:RRB -11.4

深さ優先探索

- 1:JJ を処理

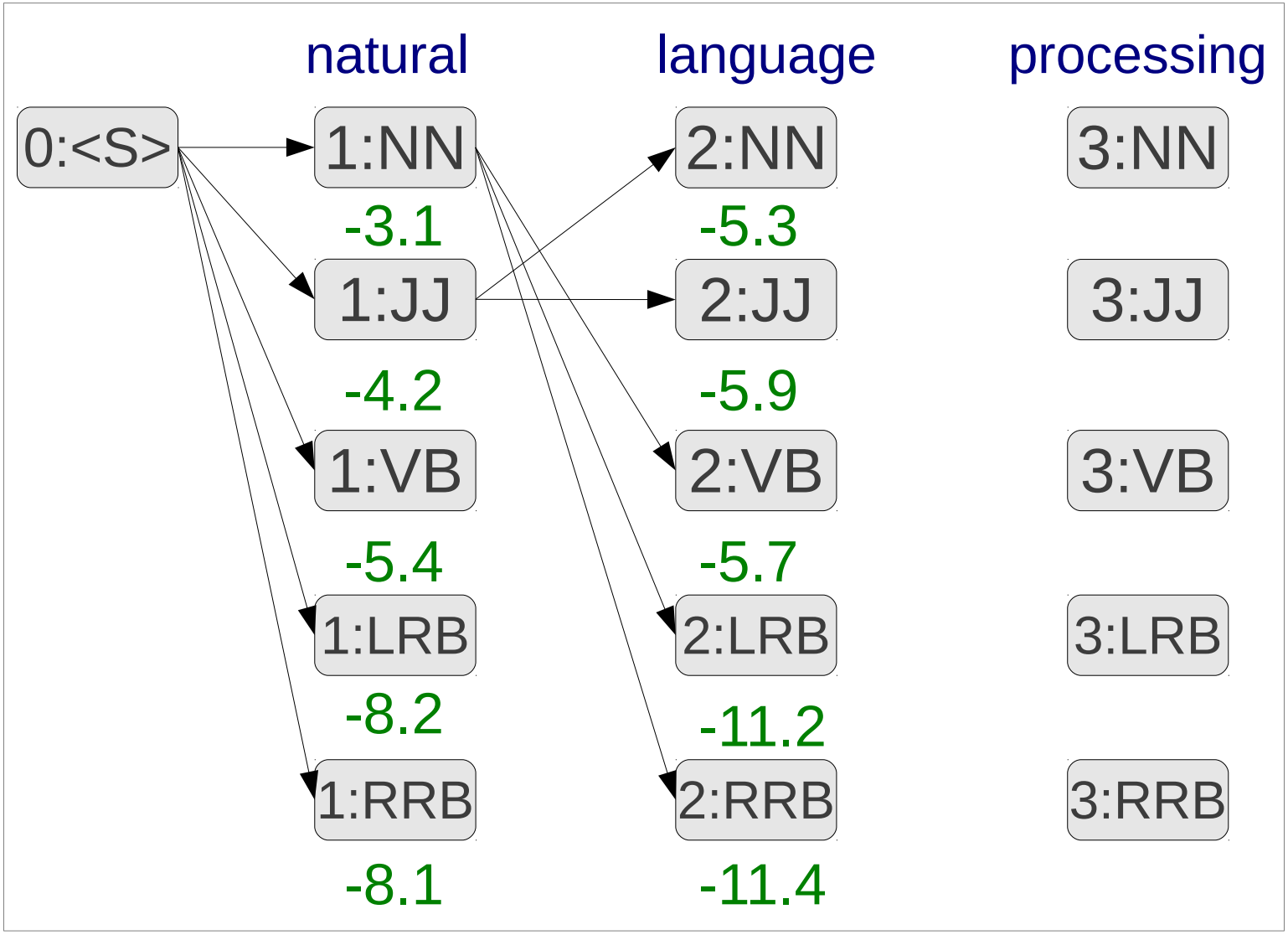


ヒープ

2:NN -5.3
1:VB -5.4
2:NN -5.5
2:VB -5.7
2:JJ -5.9
2:JJ -6.7
1:RRB -8.1
1:LRB -8.2
2:LRB -11.2
2:RRB -11.4

深さ優先探索

- 1:JJ を処理

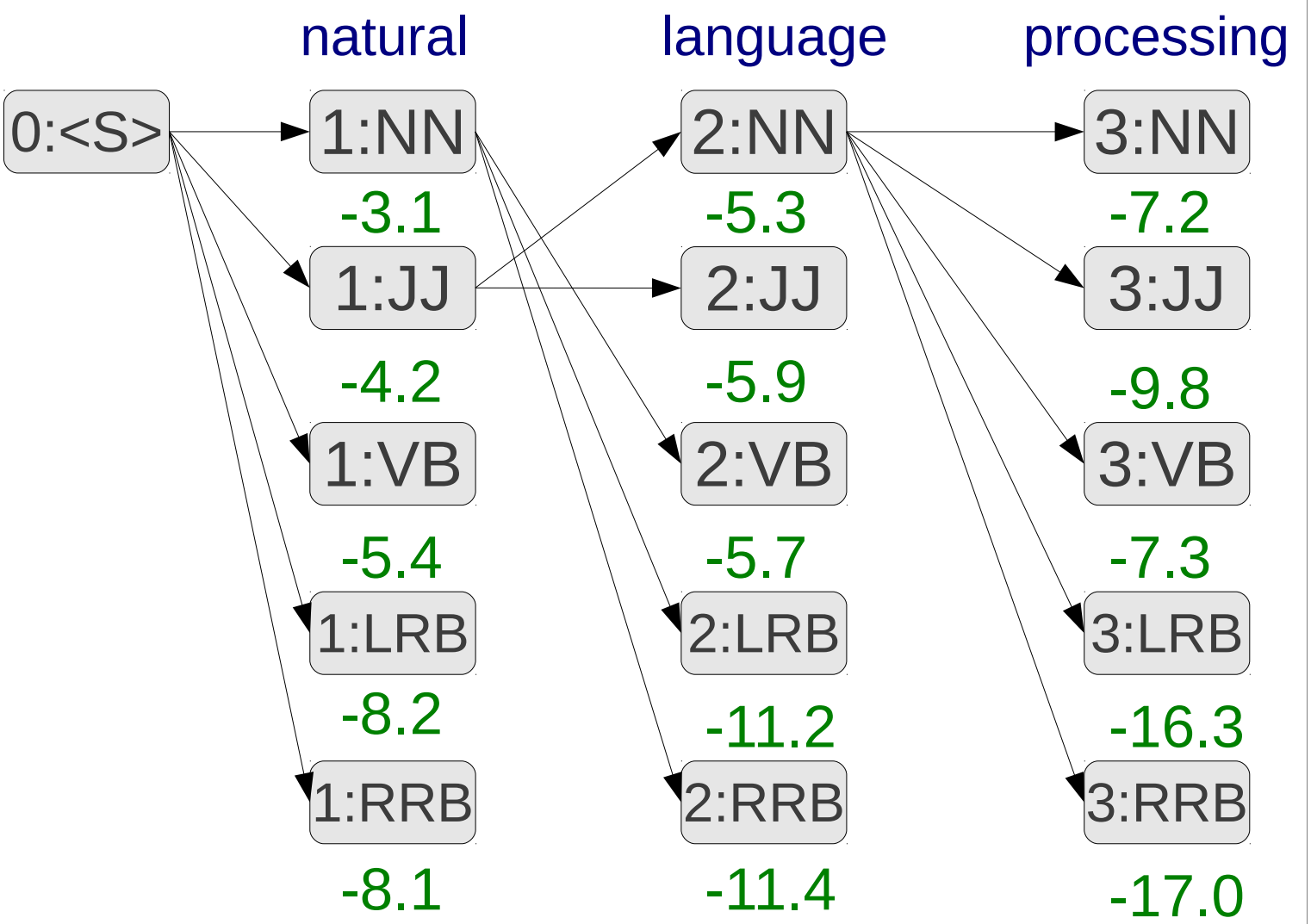


ヒープ

2:NN -5.3
1:VB -5.4
2:NN -5.5
2:VB -5.7
2:JJ -5.9
2:JJ -6.7
1:RRB -8.1
1:LRB -8.2
2:LRB -11.2
2:RRB -11.4

深さ優先探索

- 2:NN を処理

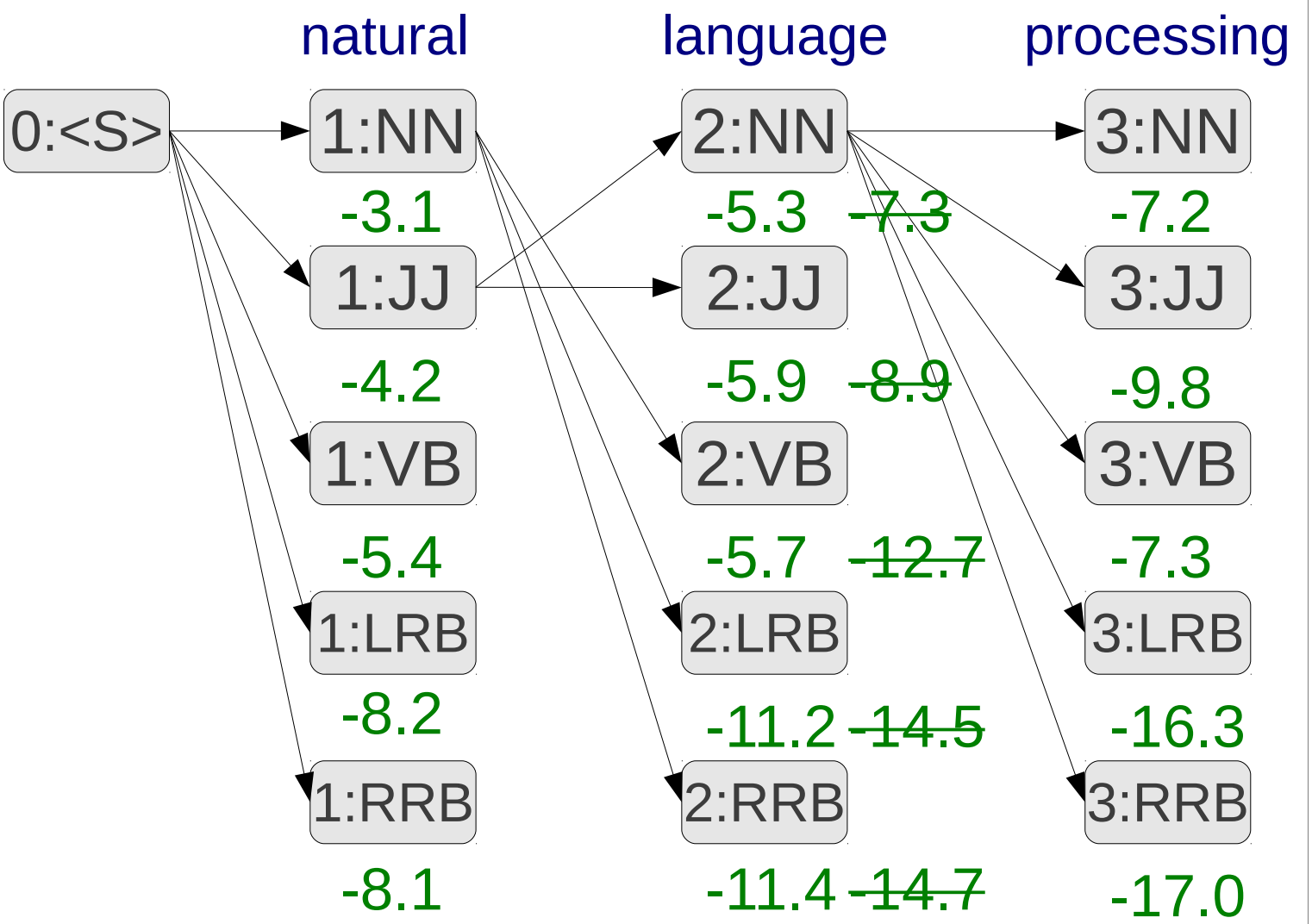


ヒープ

1:VB -5.4
2:NN -5.5
2:VB -5.7
2:JJ -5.9
2:JJ -6.7
3:NN -7.2
3:VB -7.3
1:RRB -8.1
1:LRB -8.2
3:JJ -9.8
2:LRB -11.2
...

深さ優先探索

- 1:VB を処理

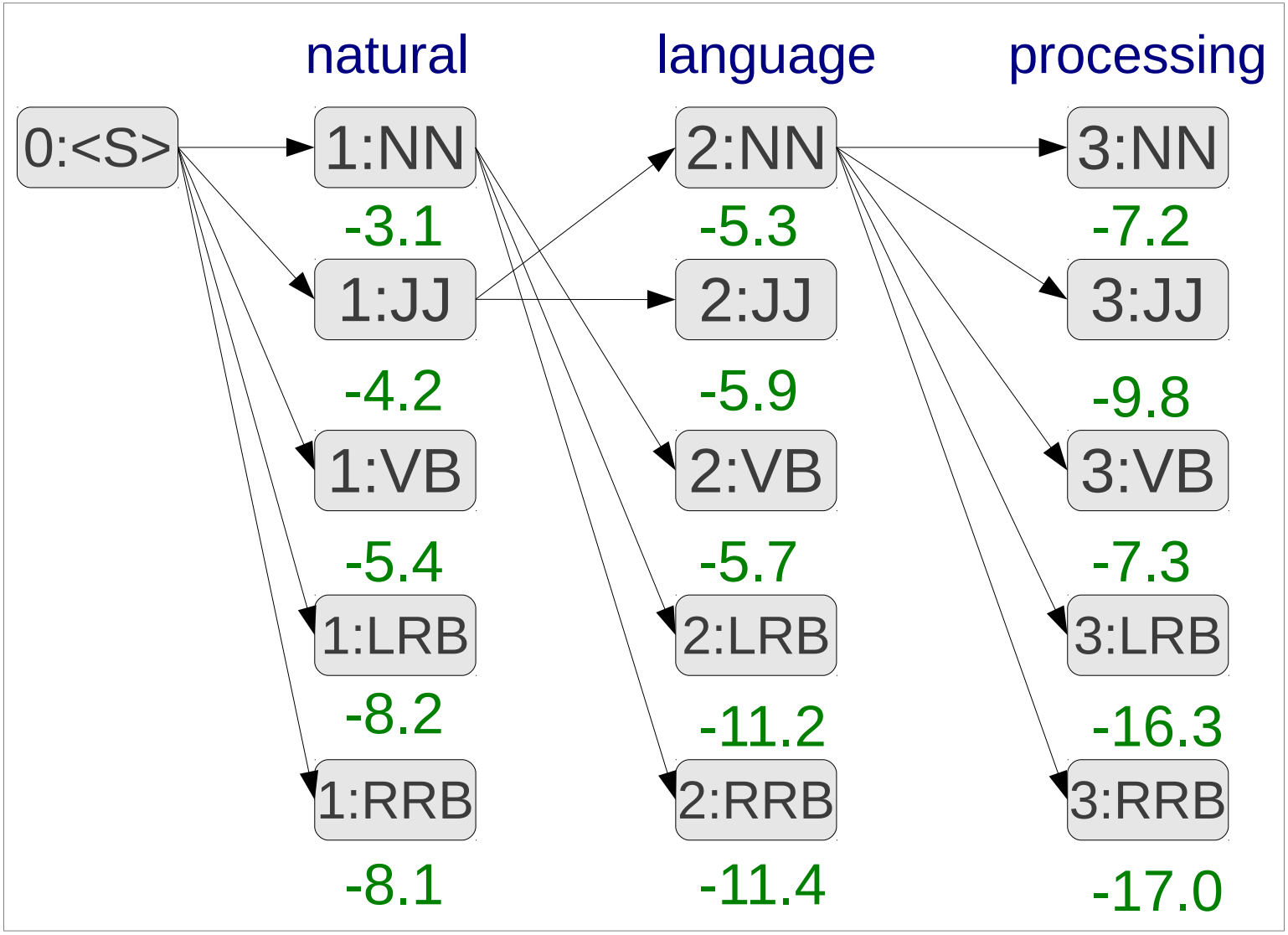


ヒープ

2:NN -5.5
2:VB -5.7
2:JJ -5.9
2:JJ -6.7
3:NN -7.2
3:VB -7.3
1:RRB -8.1
1:LRB -8.2
3:JJ -9.8
2:LRB -11.2
…

深さ優先探索

- 2:NN は処理しない（既に処理済み）



ヒープ

2:VB -5.7
2:JJ -5.9
2:JJ -6.7
3:NN -7.2
3:VB -7.3
1:RRB -8.1
1:LRB -8.2
3:JJ -9.8
2:LRB -11.2
...

問題：まだ効率に問題あり

- 長い文に弱い
- なぜ？
  - ヒント：「1:VB」を展開する必要はあったか？

A* 探索：ヒューリスティック関数を追加

- まだ未処理の単語にかかるコストも考慮
- ヒューリスティック関数：必ず実際にかかるコスト以下のコストを返す関数
- タグ付けの場合：最小生成確率の和

natural	language	processing
log(P(natural|NN)) = -2.4	log(P(lang.|NN)) = -2.4	log(P(proc.|NN)) = -2.5
log(P(natural|JJ)) = -2.0	log(P(lang.|JJ)) = -3.0	log(P(proc.|JJ)) = -3.4
log(P(natural|VB)) = -3.1	log(P(lang.|VB)) = -3.2	log(P(proc.|VB)) = -1.5
log(P(natural|LRB)) = -7.0	log(P(lang.|LRB)) = -7.9	log(P(proc.|LRB)) = -6.9
log(P(natural|RRB)) = -7.0	log(P(lang.|RRB)) = -7.9	log(P(proc.|RRB)) = -6.9

H(1+) = -5.9　　H(2+) = -3.9　　H(3+) = -1.5　　H(4+) = 0.0

A* 探索：ヒューリスティックを考慮した順位付け

- 前向きスコアとヒューリスティック関数

通常のヒープ

2:VB	F(2:VB)=-5.7	H(3+)=-1.5
2:JJ	F(2:JJ)=-5.9	H(3+)=-1.5
2:JJ	F(2:JJ)=-6.7	H(3+)=-1.5
3:NN	F(3:NN)=-7.2	H(4+)=-0.0
3:VB	F(3:VB)=-7.3	H(4+)=-0.0
1:RRB	F(1:RRB)=-8.1	H(2+)=-3.9
1:LRB	F(1:LRB)=-8.2	H(2+)=-3.9
3:JJ	F(3:JJ)=-9.8	H(4+)=-0.0
2:LRB	F(2:LRB)=-11.2	H(3+)=-1.5

A* ヒープ

3:NN	-7.2
2:VB	-7.2
3:VB	-7.3
2:JJ	-7.4
2:JJ	-8.2
3:JJ	-9.8
1:RRB	-12.0
1:LRB	-12.1
2:LRB	-12.7

演習課題

演習課題

- 実装　test-hmm-beam
- テスト
  - 入力：`test/05-{train,test}-input.txt`
  - 正解：`test/05-{train,test}-answer.txt`
- 学習　`data/wiki-en-train.norm_pos`
  実行　`data/wiki-en-test.norm`
- 評価　タグ付けの精度評価
  `script/gradepos.pl data/wiki-en-test.pos my_answer.pos`
- 比較　様々なビームサイズを比較
- チャレンジ　A* 探索の実装

Thank You!