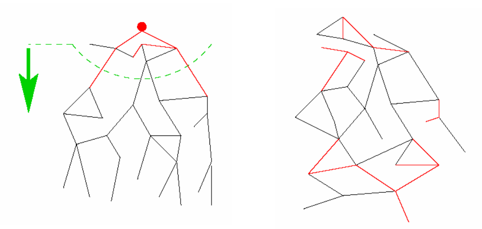
Two Different Approaches



Prim's Algorithm
Almost identical to Dijkstra's

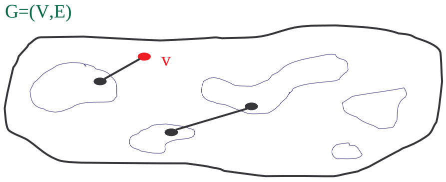
Kruskals's Algorithm
Completely different!

1

Kruskal's MST Algorithm

Idea: Grow a forest out of edges that do not create a cycle. Pick an ***edge with the smallest weight.***

G=(V,E)



2

Kruskal's Algorithm for MST

An *edge-based* greedy algorithm

Builds MST by greedily adding edges

1. Initialize with
   - empty MST
   - all vertices marked unconnected
   - all edges unmarked
2. While there are still unmarked edges
   a. Pick the lowest cost edge `(u,v)` and mark it
   b. If `u` and `v` are not already connected, add `(u,v)` to the MST and mark `u` and `v` as connected to each other

Doesn't it sound familiar?

3

Kruskal Pseudo Code

```
void Graph::kruskal(){
  int edgesAccepted = 0;
  DisjSet s(NUM_VERTICES);

  while (edgesAccepted < NUM_VERTICES – 1){
    e = smallest weight edge not deleted yet;
    // edge e = (u, v)
    uset = s.find(u);
    vset = s.find(v);
    if (uset != vset){
      edgesAccepted++;
      s.unionSets(uset, vset);
    }
  }
}
```

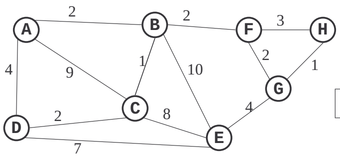
Complexity? (arrow to `while (edgesAccepted < NUM_VERTICES – 1)`)

Complexity? (arrow to `vset = s.find(v);`)

Complexity? (arrow to `s.unionSets(uset, vset);`)

4

Find MST using Kruskal's



Total Cost:

- Now find the MST using Prim's method.
- Under what conditions will these methods give the same result?

5

Kruskal's Algorithm: Correctness

It clearly generates a spanning tree. Call it T_K.

Suppose T_K is *not* minimum:

Pick another spanning tree T_{min} with *lower cost* than T_K

Pick the smallest edge $e_1=(u,v)$ in T_K that is **not** in T_{min}

T_{min} already has a path p in T_{min} from u to v

$\Rightarrow$ Adding e_1 to T_{min} will create a cycle in T_{min}

Pick an edge e_2 in p that Kruskal's algorithm considered *after* adding e_1 (must exist: u and v unconnected when e_1 considered)

$\Rightarrow$ cost(e_2) $\geq$ cost(e_1)

$\Rightarrow$ can replace e_2 with e_1 in T_{min} without increasing cost!

Keep doing this until T_{min} is identical to T_K

$\Rightarrow$ T_K must also be minimal – contradiction!

6

Return to Dynamic Programming

- Recall that dynamic programming is a technique that reuses computed values of intermediate computations:
- Fib(n) = Fib(n-1) + Fib(n-2)
- A classic description of a computation that is suitable for dynamic programming is the form

  $f(i, k) = \min_j (f(i, j) + f(j, k))$

7

Context Free Grammar

- A grammar G=(T, N, S, P) where
  - T is a set of terminals, e.g. T={a, b, n, s}
  - N is a set of non-terminals, e.g. N={S, A, B}
  - S is the start symbol
  - P is a set of productions of the form N $\rightarrow$ string

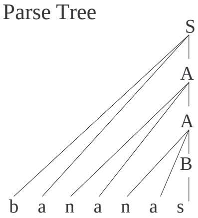
{ S $\rightarrow$ baA
A $\rightarrow$ naA
A $\rightarrow$ B
B $\rightarrow$ s
B $\rightarrow$ ε}

8

A Generation

S => baA => banaA => bananaA => bananaB => bananas

- Parse Tree

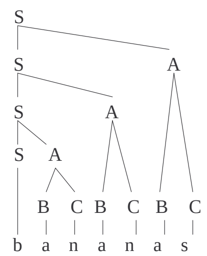


{ S $\rightarrow$ baA
A $\rightarrow$ naA
A $\rightarrow$ B
B $\rightarrow$ s
B $\rightarrow$ ε}

9

Alternative Grammar

- There are many ways to express strings by cfgs



{ S $\rightarrow$ SA
S $\rightarrow$ b
A $\rightarrow$ BC
B $\rightarrow$ a
C $\rightarrow$ n
A $\rightarrow$ a
C $\rightarrow$ s}

10

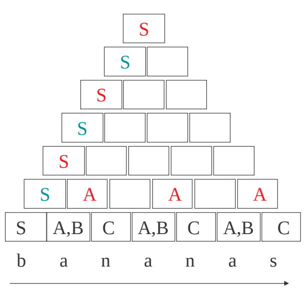
Parse by Reversing Arrows

bananas =>
=> Sananas *using* S $\rightarrow$ b
=> SBnanas *using* B $\rightarrow$ a
=> SBCanas *using* C $\rightarrow$ n
=> SAanas using A $\rightarrow$ BC
=> Sanas *using* S $\rightarrow$ SA
=> SAnas *using* A $\rightarrow$ a
=> Snas *using* S $\rightarrow$ SA
=> ?

{ S $\rightarrow$ SA
S $\rightarrow$ b
A $\rightarrow$ BC
B $\rightarrow$ a
C $\rightarrow$ n
A $\rightarrow$ a
C $\rightarrow$ s}

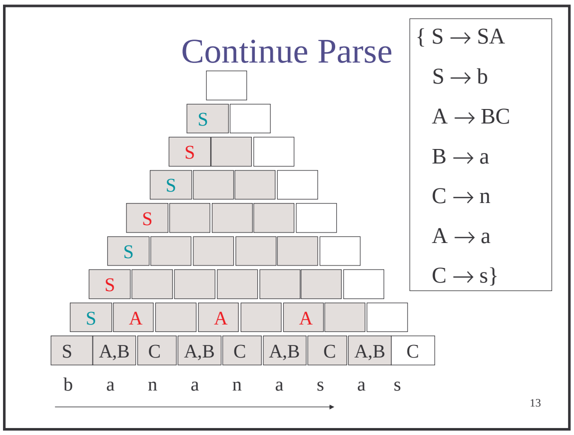
11

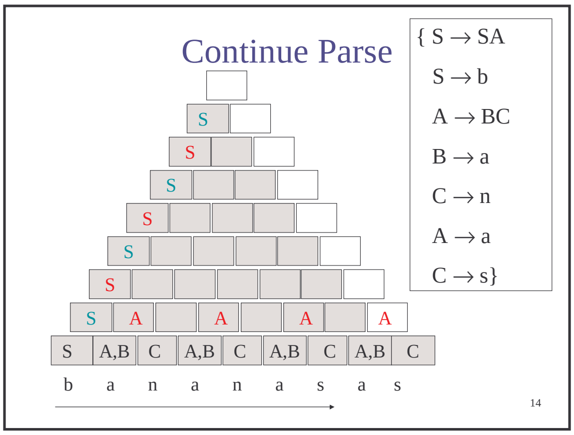
Parse By Dynamic Programming

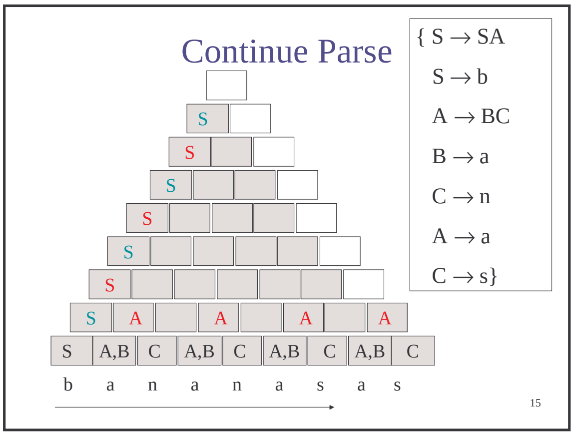


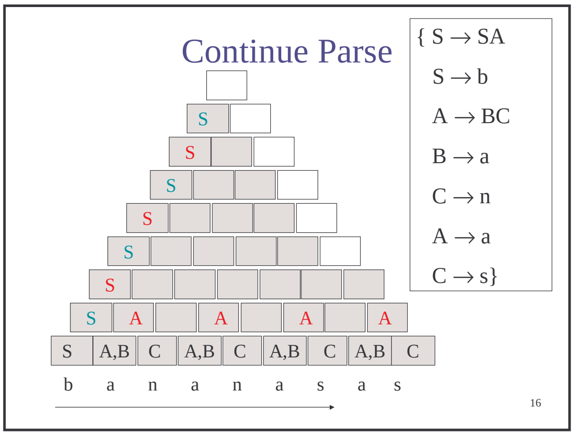
{ S $\rightarrow$ SA
S $\rightarrow$ b
A $\rightarrow$ BC
B $\rightarrow$ a
C $\rightarrow$ n
A $\rightarrow$ a
C $\rightarrow$ s}

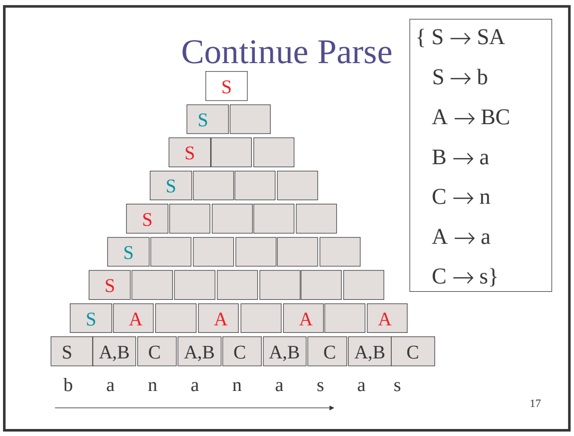
12


Continue Parse
{ S → SA
S → b
A → BC
B → a
C → n
A → a
C → s}
S
S
S
S
S
S
S A A A
S A,B C A,B C A,B C A,B C
b a n a n a s a s
13


Continue Parse
{ S → SA
S → b
A → BC
B → a
C → n
A → a
C → s}
S
S
S
S
S
S
S A A A A
S A,B C A,B C A,B C A,B C
b a n a n a s a s
14


Continue Parse
{ S → SA
S → b
A → BC
B → a
C → n
A → a
C → s}
S
S
S
S
S
S
S A A A A
S A,B C A,B C A,B C A,B C
b a n a n a s a s
15


Continue Parse
{ S → SA
S → b
A → BC
B → a
C → n
A → a
C → s}
S
S
S
S
S
S
S A A A A
S A,B C A,B C A,B C A,B C
b a n a n a s a s
16


Continue Parse
{ S → SA
S → b
A → BC
B → a
C → n
A → a
C → s}
S
S
S
S
S
S
S
S A A A A
S A,B C A,B C A,B C A,B C
b a n a n a s a s
17