Best-First Search
Minimizing Space or Time

RBFS

Take more space than IDA*

Take less time than IDA*

RBFS general properties

- Similar to A* algorithm developed for heuristic search

RBFS general properties – 2

- Similar to A* algorithm developed for heuristic search
  - **Both are recursive in the same sense**

RBFS general properties – 3

- Similar to A* algorithm developed for heuristic search
  - **Both are recursive in the same sense**

- Difference between A* and RBFS

RBFS general properties – 3

- Similar to A* algorithm developed for heuristic search
  - **Both are recursive in the same sense**

- Difference between A* and RBFS
  - **A* keeps in memory all of the already generated nodes**

RBFS general properties – 4

- Similar to A* algorithm developed for heuristic search
  - **Both are recursive in the same sense**

- Difference between A* and RBFS
  - **A* keeps in memory all of the already generated nodes**
  - **RBFS only keeps the current search path and the sibling nodes along the path**

RBFS space – 2

» When does RBFS suspend the search of a subtree?

RBFS space – 3

» When does RBFS suspend the search of a subtree?

> When it no longer looks the best

RBFS space – 3

- When does RBFS suspend the search of a subtree?
  - When it no longer looks the best

- What does it do when a subtree is suspended?

RBFS space – 3

» When does RBFS suspend the search of a subtree?

> When it no longer looks the best

» What does it do when a subtree is suspended?

> It forgets the subtree to save space

RBFS space – 4

- When does RBFS suspend the search of a subtree?
  - When it no longer looks the best
- What does it do when a subtree is suspended?
  - It forgets the subtree to save space
- What is the space complexity?

RBFS space – 5

» When does RBFS suspend the search of a subtree?

> When it no longer looks the best

» What does it do when a subtree is suspended?

> It forgets the subtree to save space

» What is the space complexity?

> Linear with depth of the search

RBFS space – 6

- When does RBFS suspend the search of a subtree?
  - When it no longer looks the best
- What does it do when a subtree is suspended?
  - It forgets the subtree to save space
- What is the space complexity?
  - Linear with depth of the search
    - Same as IDA*

RBFS memory

- When RBFS suspends searching a subtree, what does it remember?

RBFS memory – 2

» When RBFS suspends searching a subtree, what does it remember?

> An updated f-value of the root of the subtree

Updated f-values

» How does RBFS update the f-values?

Updated f-values – 2

» How does RBFS update the f-values?

> Backing up the f-values in the same way as A* does

f-value notation

- Static f-value
  - **f(N)**
    - **Value returned by the evaluation function**
    - **Always the same**

f-value notation – 2

- Static f-value
  - **f(N)**
    - **Value returned by the evaluation function**
    - **Always the same**

- Backed-up value
  - **F(N)**
    - **Changes during the search**
      - **Depends upon descendants of N**

F(N) definition

- RBFS backs up f-values in the same way as A*
  - **How is F(N) defined?**

F(N) definition – 2

- RBFS backs up f-values in the same way as A*
  - **How is F(N) defined?**
    - **If N has never been expanded?**

F(N) definition – 3

- RBFS backs up f-values in the same way as A*
  - **How is F(N) defined?**
    - **If N has never been expanded?**
      - **F(N) = f(N)**

F(N) definition – 4

- RBFS backs up f-values in the same way as A*
  - **How is F(N) defined?**
    - **If N has never been expanded?**
      - **F(N) = f(N)**
    - **If N has been expanded?**

F(N) definition – 5

- RBFS backs up f-values in the same way as A*

  - **How is F(N) defined?**

    - **If N has never been expanded?**
      - **F(N) = f(N)**

    - **If N has been expanded?**
      - **F(N) = min (F (S_j))**
      - **Where S_j are the subtrees of N**

Subtree exploration

» How does RBFS explore subtrees?

Subtree exploration – 2

» How does RBFS explore subtrees?

> As in A*, within a given f-bound

Subtree exploration – 3

» How does RBFS explore subtrees?

> As in A*, within a given f-bound

» How is the bound determined?

RBFS subtree exploration – 4

» How does RBFS explore subtrees?

> As in A*, within a given f-bound

» How is the bound determined?

> From the F-values of the siblings along the current search path

Subtree exploration – 5

» How does RBFS explore subtrees?

> As in A*, within a given f-bound

» How is the bound determined?

> From the F-values of the siblings along the current search path

> The smallest F-value

– The closest competitor

Subtree exploration – 6

- Suppose N is currently the best node

Subtree exploration – 7

- Suppose N is currently the best node
  - > **N is expanded**

Subtree exploration – 8

- Suppose N is currently the best node
  - > **N is expanded**
  - > **N's children are expanded**

Subtree exploration – 9

- Suppose N is currently the best node
  - > **N is expanded**
  - > **N's children are expanded**

  - » **Until when?**

Subtree exploration – 10

- Suppose N is currently the best node
  - **N is expanded**
  - **N's children are expanded**

  - **Until when?**
    - **F(N) > Bound**

Subtree exploration – 10

- Suppose N is currently the best node
  - > **N is expanded**
  - > **N's children are expanded**

  - » **Until when?**
    - > **F(N) > Bound**

  - » **Then what happens?**

Subtree exploration – 11

- Suppose N is currently the best node
  - > **N is expanded**
  - > **N's children are expanded**

  - » **Until when?**
    - > **F(N) > Bound**

  - » **Then what happens?**
    - > **Nodes below N are forgotten**

Subtree exploration – 12

- Suppose N is currently the best node
  - > **N is expanded**
  - > **N's children are expanded**

  - » **Until when?**
    - > **F(N) > Bound**

  - » **Then what happens?**
    - > **Nodes below N are forgotten**
    - > **N's F-value is updated**

Subtree exploration – 13

- Suppose N is currently the best node
  - > **N is expanded**
  - > **N's children are expanded**

  - » **Until when?**
    - > **F(N) > Bound**

  - » **Then what happens?**
    - > **Nodes below N are forgotten**
    - > **N's F-value is updated**
    - > **RBFS selects which node to expand next**

F-value inheritance

- F-values can be inherited from a node's parents

F-value inheritance – 2

- F-values can be inherited from a node's parents
- Let N be a node about to be expanded

F-value inheritance – 3

- F-values can be inherited from a node's parents
- Let N be a node about to be expanded
  - **If F(N) > f(N) then N had already been expanded**

F-value inheritance – 4

- F-values can be inherited from a node's parents
- Let N be a node about to be expanded
  - **If F(N) > f(N) then N had already been expanded**
  - **F(N) was determined from N's children**

F-value inheritance – 5

- F-values can be inherited from a node's parents
- Let N be a node about to be expanded
  - **If F(N) > f(N) then N had already been expanded**
  - **F(N) was determined from N's children**
  - **Children have been removed from memory**

F-value inheritance – 6

- F-values can be inherited from a node's parents
- Let N be a node about to be expanded
  - **If F(N) > f(N) then N had already been expanded**
  - **F(N) was determined from N's children**
  - **Children have been removed from memory**
- Suppose a child N_k of N is generated again

F-value inheritance – 7

- F-values can be inherited from a node's parents
- Let N be a node about to be expanded
  - **If F(N) > f(N) then N had already been expanded**
  - **F(N) was determined from N's children**
  - **Children have been removed from memory**
- Suppose a child N_k of N is generated again
  - **Compute $f(N_k)$**

F-value inheritance – 8

- F-values can be inherited from a node's parents
- Let N be a node about to be expanded
  - **If F(N) > f(N) then N had already been expanded**
  - **F(N) was determined from N's children**
  - **Children have been removed from memory**
- Suppose a child N_k of N is generated again
  - **Compute $F(N_k)$**
  - **$F(N_k) = \max (F(N) , f(N_k))$**

F-value inheritance – 9

- F-values can be inherited from a node's parents
- Let N be a node about to be expanded
  - **If F(N) > f(N) then N had already been expanded**
  - **F(N) was determined from N's children**
  - **Children have been removed from memory**
- Suppose a child N_k of N is generated again
  - **Compute $f(N_k)$**
  - **$F(N_k) = \max (F(N) , f(N_k))$**
    - **N_k's F-value can be inherited from N**

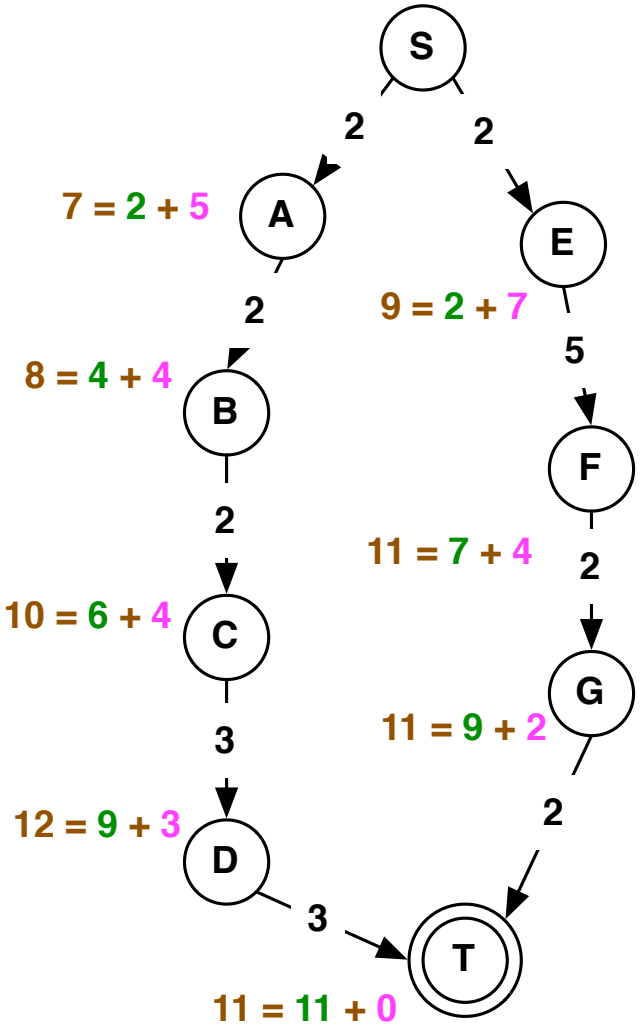
F-value inheritance – 10

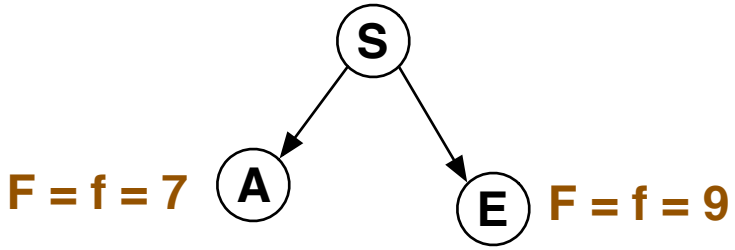
- F-values can be inherited from a node's parents
- Let N be a node about to be expanded
  - **If F(N) > f(N) then N had already been expanded**
  - **F(N) was determined from N's children**
  - **Children have been removed from memory**
- Suppose a child N_k of N is generated again
  - **Compute $f(N_k)$**
  - **$F(N_k) = \max (F(N) , f(N_k))$**
    - **N_k's F-value can be inherited from N**
      - **N_k was generated earlier**

F-value inheritance – 11

- F-values can be inherited from a node's parents
- Let N be a node about to be expanded
  - **If F(N) > f(N) then N had already been expanded**
  - **F(N) was determined from N's children**
  - **Children have been removed from memory**
- Suppose a child N_k of N is generated again
  - **Compute $f(N_k)$**
  - **$F(N_k) = \max (F(N) , f(N_k))$**
    - **N_k's F-value can be inherited from N**
      - **N_k was generated earlier**
      - **$F(N_k)$ was ≥ F(N), otherwise F(N) would be smaller**

Fig 12.2 snapshots



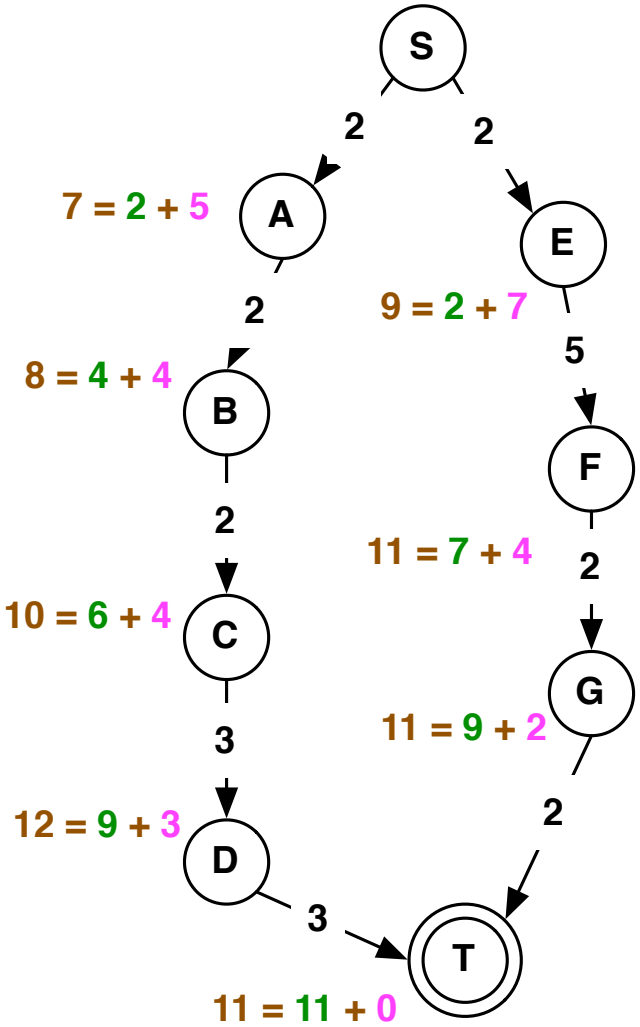


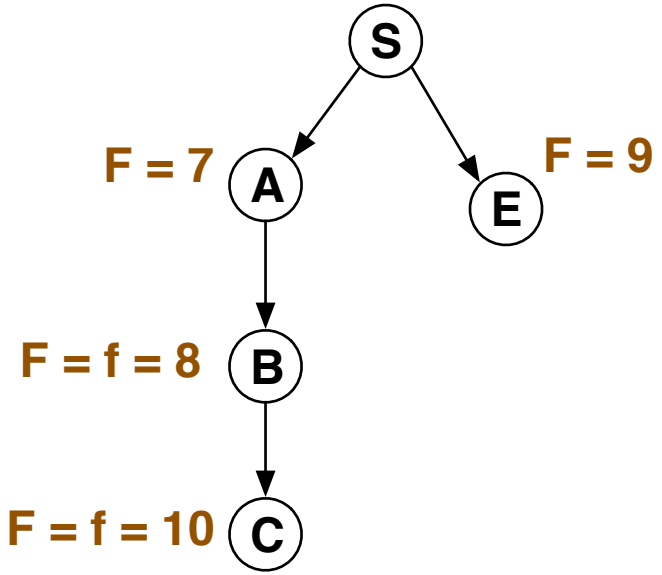
S is expanded

A is found to be the best child

f(n) in mocha = g(n) in clover + h(n) in magenta

Fig 12.2 snapshots – 2





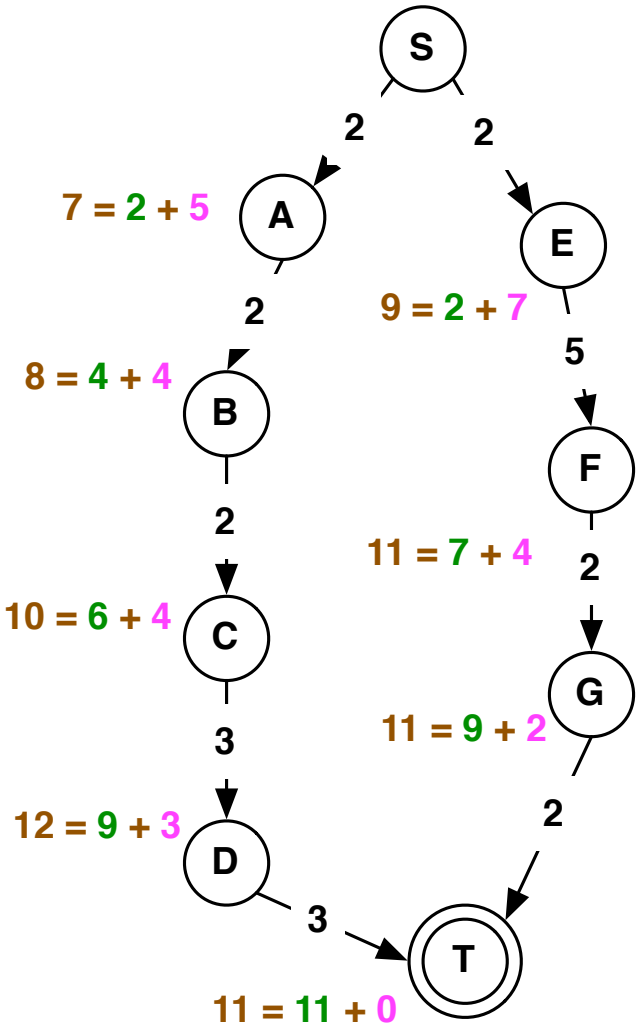
A is expanded with bound 9

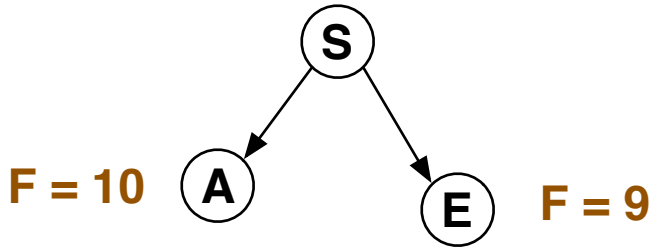
C has F-value 10

Stop expansion, backup F value

f(n) in mocha = g(n) in clover + h(n) in magenta

Fig 12.2 snapshots – 3





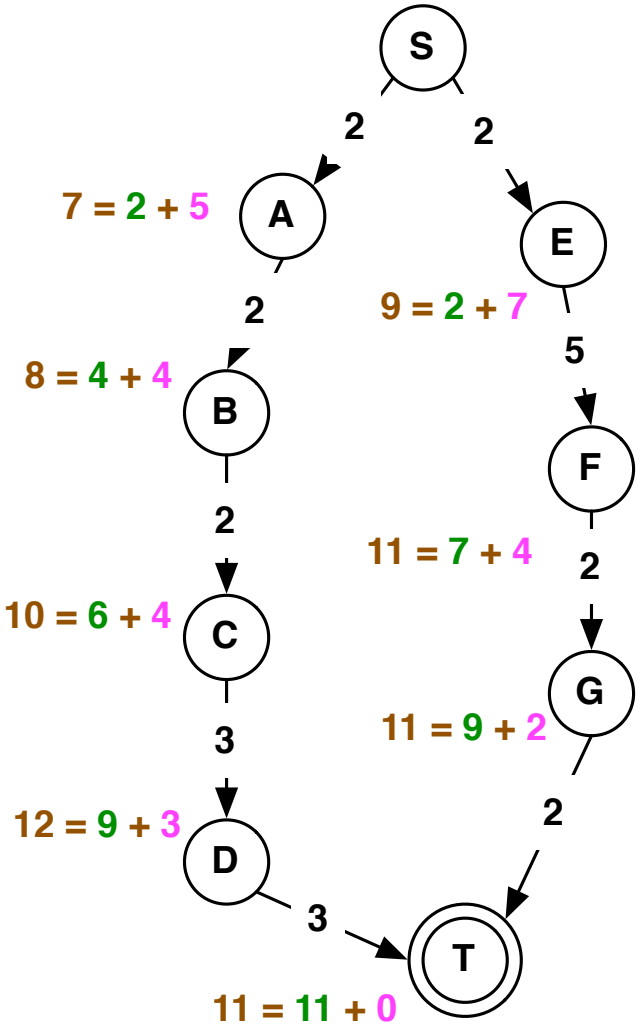
Forget expansion from A

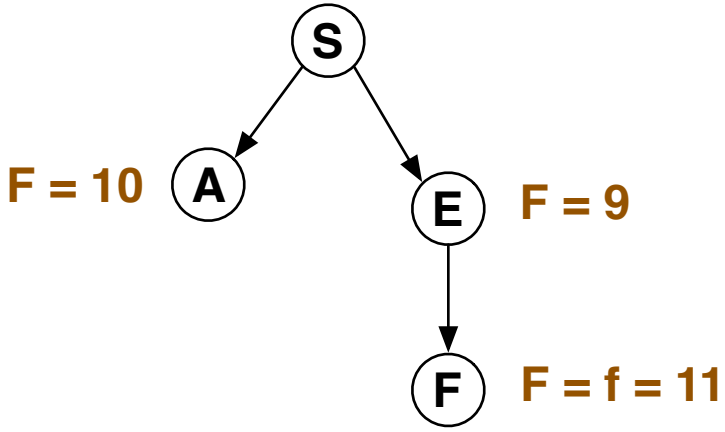
A has backed up F value 10

E is best to expand next

f(n) in mocha = g(n) in clover + h(n) in magenta

Fig 12.2 snapshots – 4





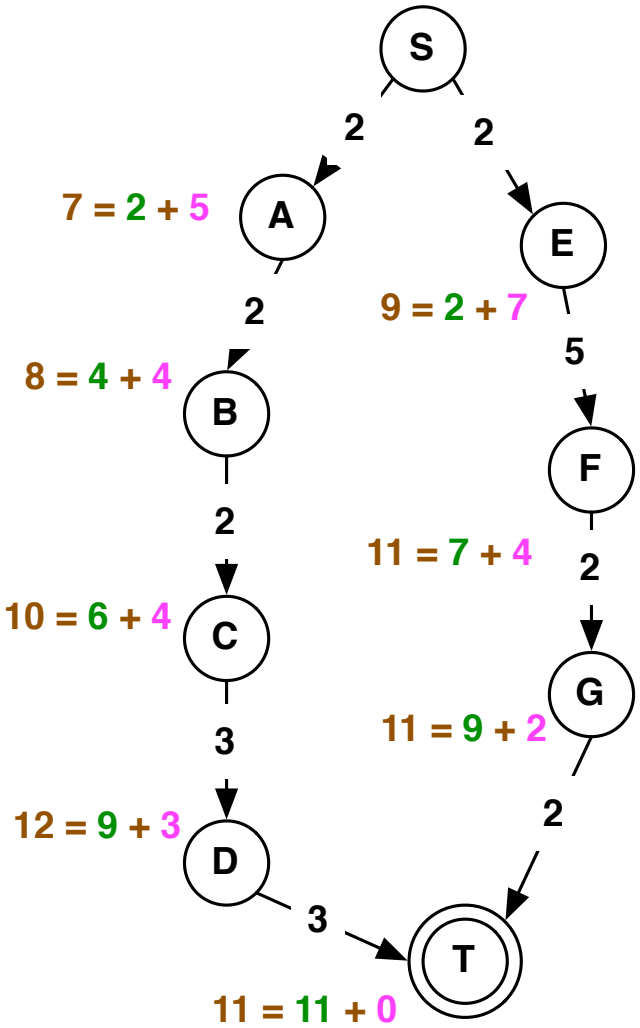
E is expanded with bound 10

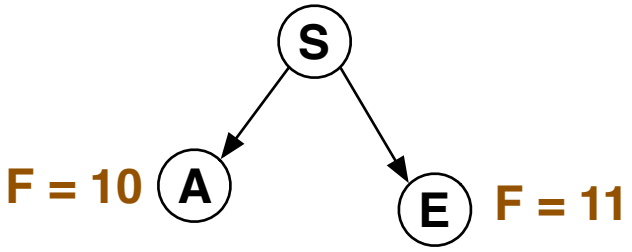
F has F-value 11

Stop expansion, backup F value

f(n) in mocha = g(n) in clover + h(n) in magenta

Fig 12.2 snapshots – 5





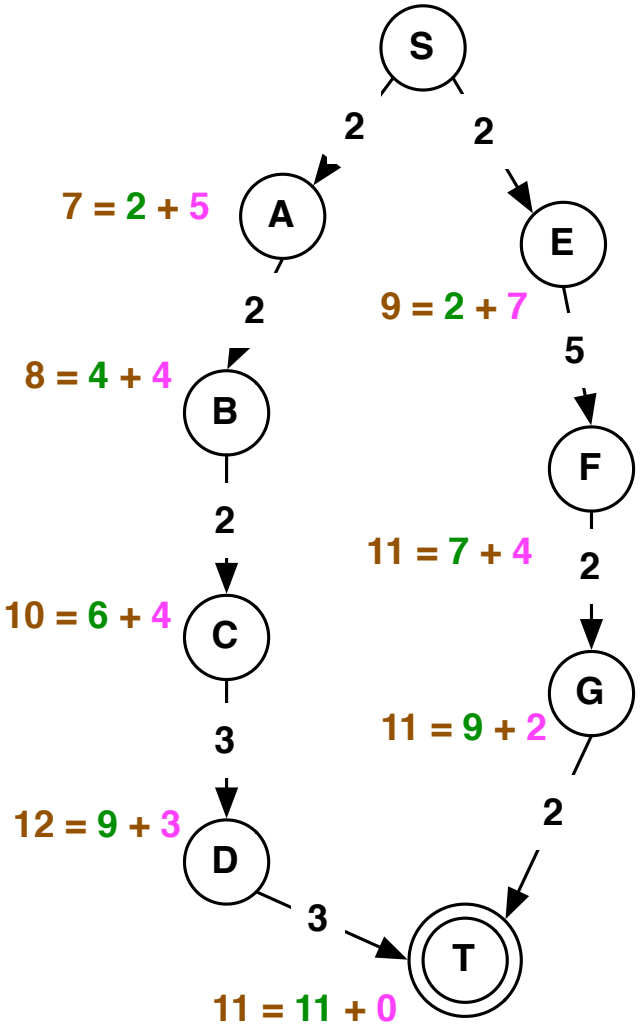
Forget expansion from E

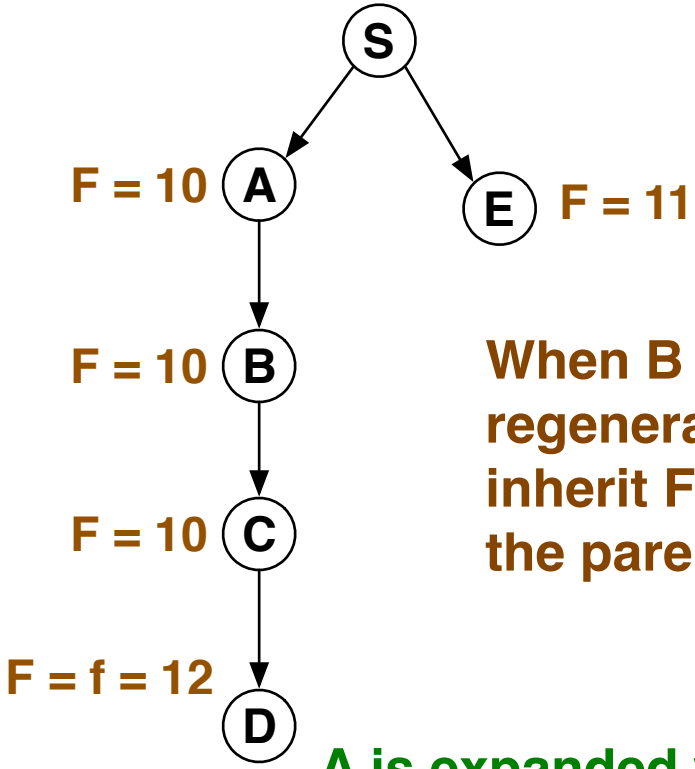
E has backed up F value 11

A is best to expand next

f(n) in mocha = g(n) in clover + h(n) in magenta

Fig 12.2 snapshots – 6





When B and C are regenerated, they inherit F value 10 from the parent

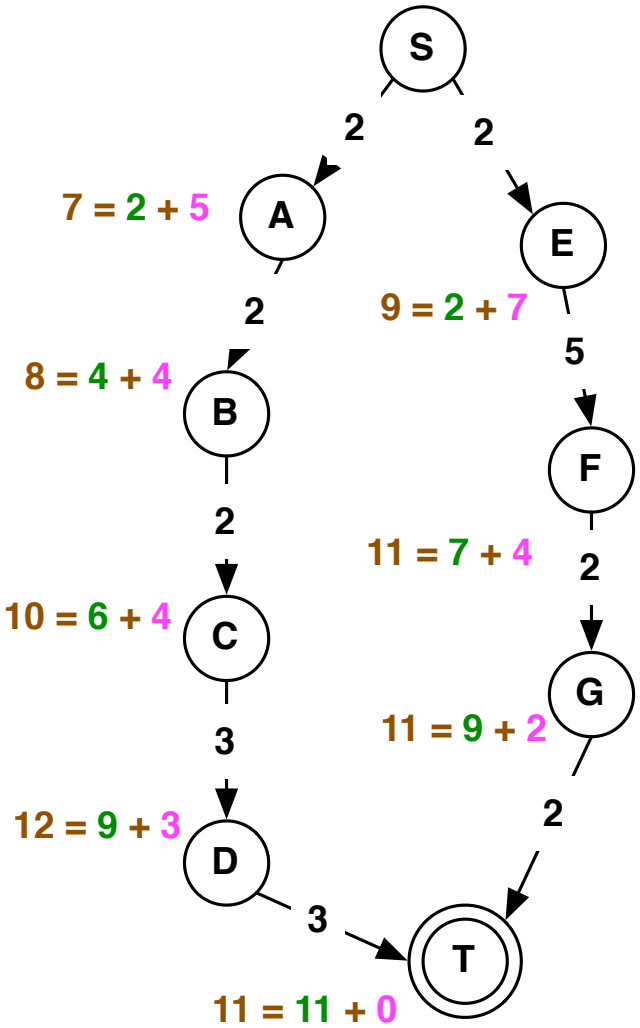
A is expanded with bound 11

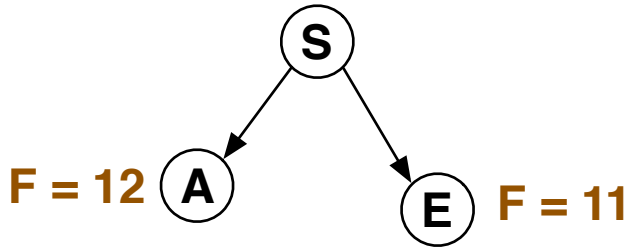
D has F-value 12

Stop expansion, backup F value

f(n) in mocha = g(n) in clover + h(n) in magenta

Fig 12.2 snapshots – 7





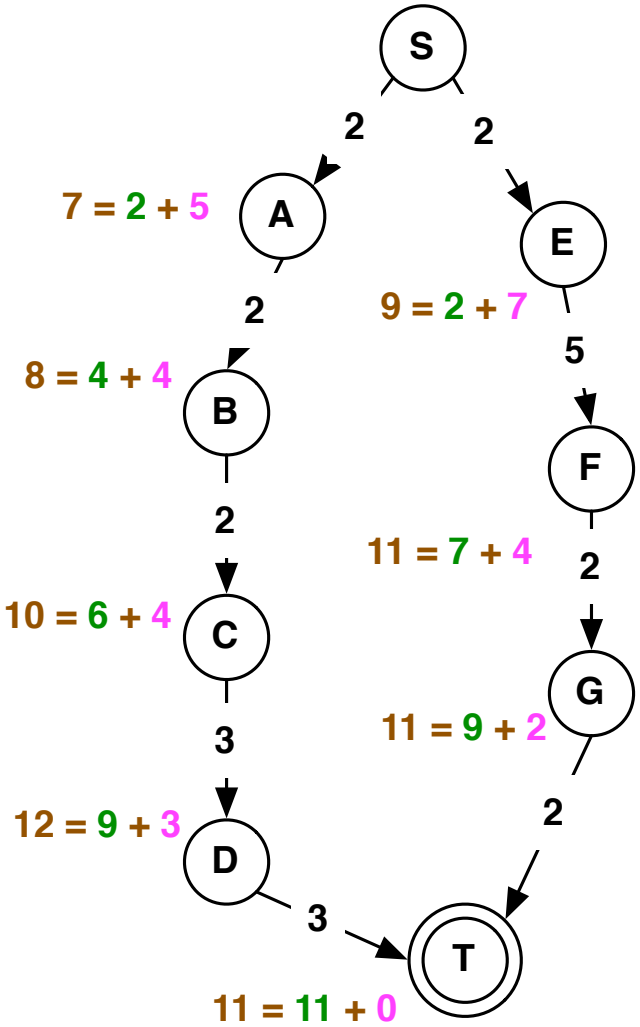
Forget expansion from A

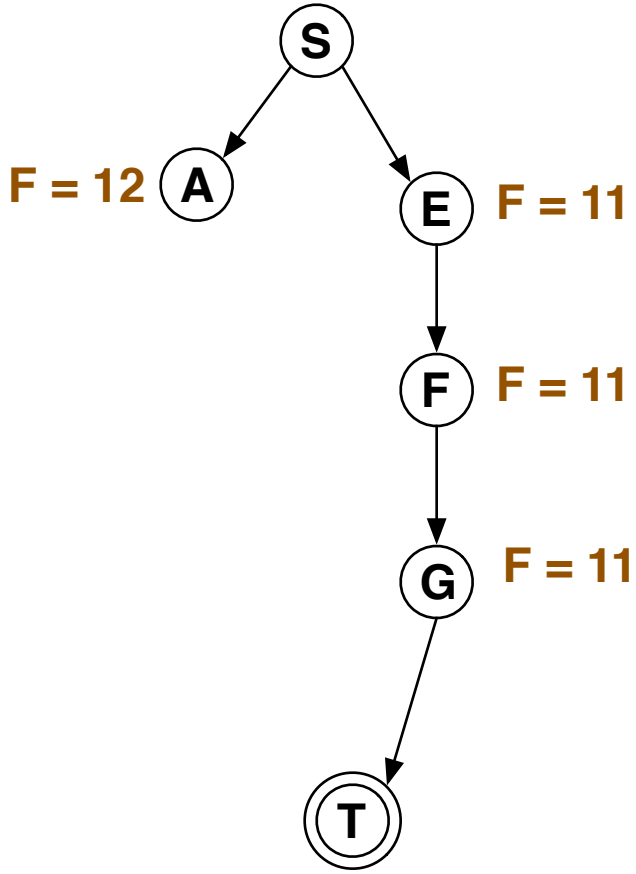
A has backed up F value 12

E is best to expand next

f(n) in mocha = g(n) in clover + h(n) in magenta

Fig 12.2 snapshots – 8





f(n) in mocha = g(n) in clover + h(n) in magenta

E is expanded with bound 12

Reach goal, search ends

Algorithm

```
function NewF (N, F(N), Bound)
   if F(N) > Bound then NewF := F(N)
   else if goal(N) then exit search with success
   else if N has no children then NewF := infinity     – dead end
   else for each child N_k of N do
          if f(N) < F(N) then F(N_k) := max( F(N), f(N_k))
                         else F(N_k) := f(N_k)
          sort children N_k in increasing order of F-value
          while F(N_1) ≤ Bound and F(N_1) < infinity do
              Bound1 := min ( Bound, F-value of sibling N_1)
              F(N_1) := NewF (N_1, F(N_1), Bound1)
              reorder nodes  N_1, N_2 , … according to new F(N_1)
             end
          end
          NewF := F(N_1)
```

Summary RBFS properties

- Space complexity
  - **Linear in depth of search**
- Cost of minimizing space
  - **Regenerate previously generated nodes**
    - **Overhead substantially less than IDA***
- Expands nodes in best-order
  - **Like A*, even with non-monotonic f-function**
  - **Unlike IDA*, which requires a monotonic f-function**