Търсене в пространство на състояния

Неинформирано търсене на път до определена цел

Пространство на състояния
STATE SPACE

Състояние е едно описание (формулировка) на задачата в процеса на нейното решаване

Пространството на състояния представя проблема чрез граф

- **възлите** съответстват на състоянията (частични решения на задачата).
- **дъгите** съответстват на операторите или стъпките в

Пространство на състояния е съвкупността от всички възможни състояния, които могат да се получат от дадено начално състояние.

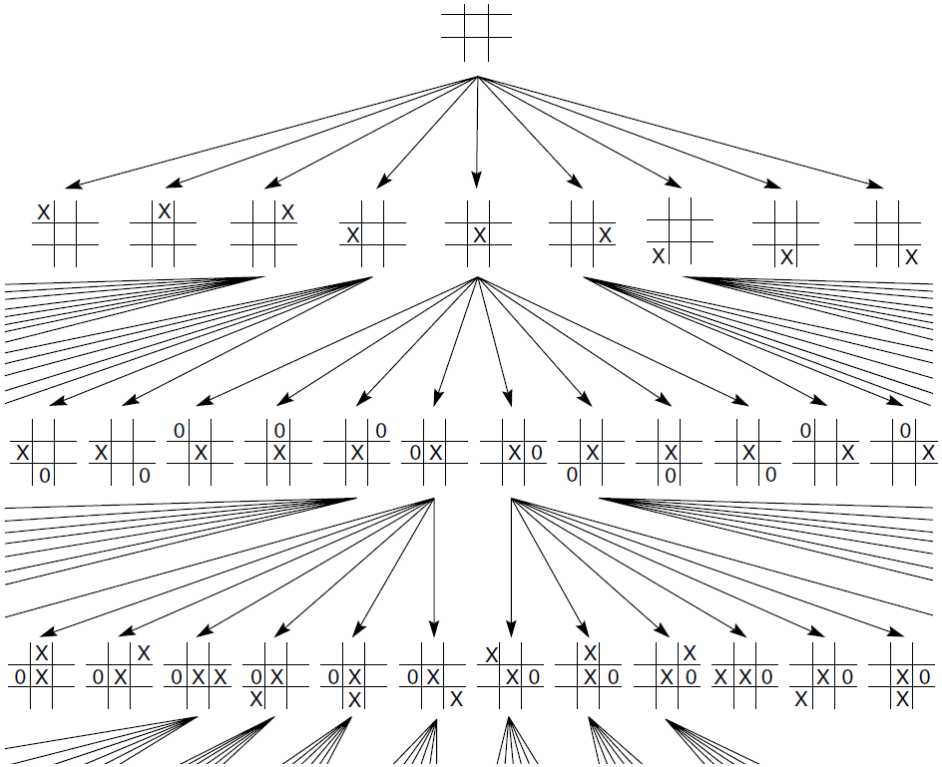
Пример- Морски шах

Пространството на състоянията съдържа всички различни конфигурации.

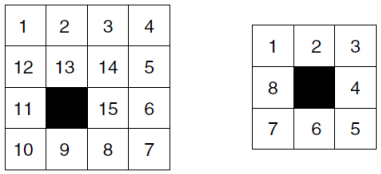
Дъгите представляват позволените ходове на играта. Състоянията се генерират, като се поставят X или O на празно място.

Пространството на състоянията е ориентиран ацикличен граф

(някои състояния могат да бъдат достигнати по различни пътища).



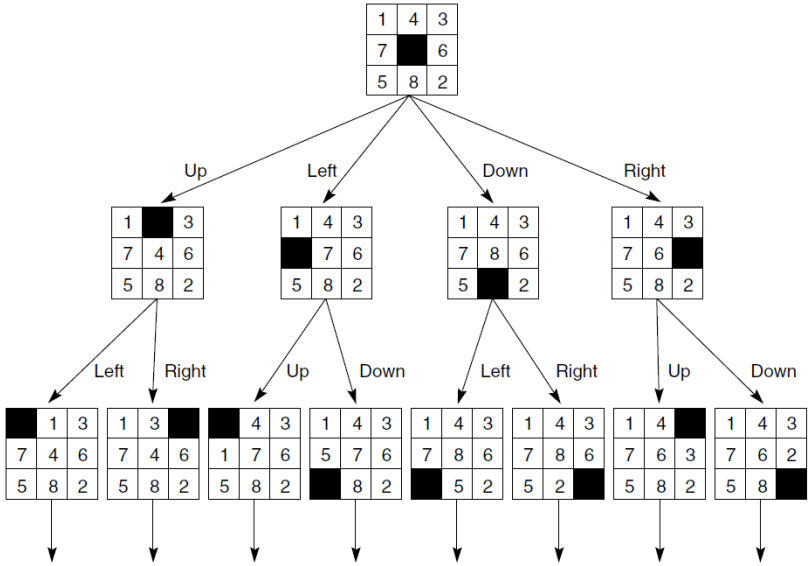
Пример: 8-те плъзгащи се плочки
Sliding puzzle

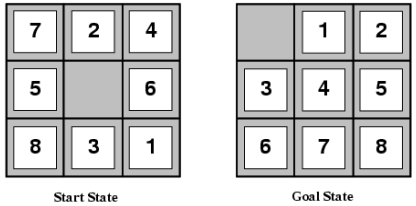


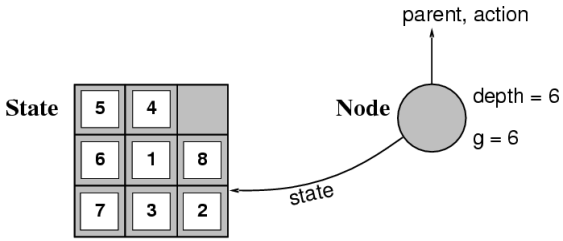
15-puzzle 8-puzzle

Състояния: разположения (конфигурации) на 8-те плочки
Оператори: преместване на празната плочка (left, right, up, down)
Проверка за достигане на целта: проверка за съвпадение с даденото целево състояние

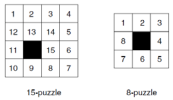
Цена на пътя: 1 за всяко преместване на празната плочка

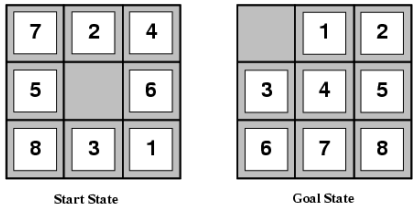


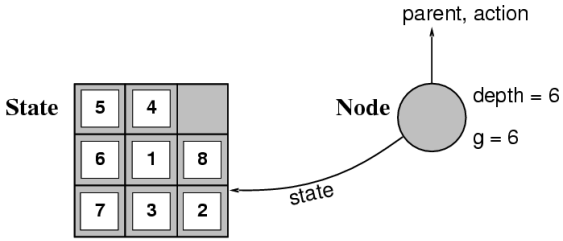


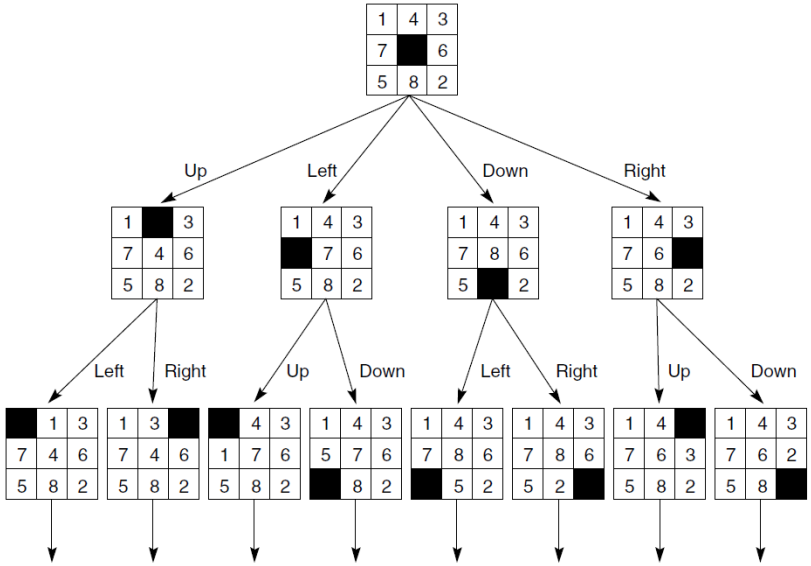


Пример: 8 плъзгащи се плочки
Sliding puzzle









Sliding puzzle

move the blank up ↑

move the blank right →

move the blank down ↓

move the blank left ←

Може да се представят с преместване на празната плочка, което опростява правилата за преминаване от едно състояние към друго.

пространството на състоянията е с възможни цикли.

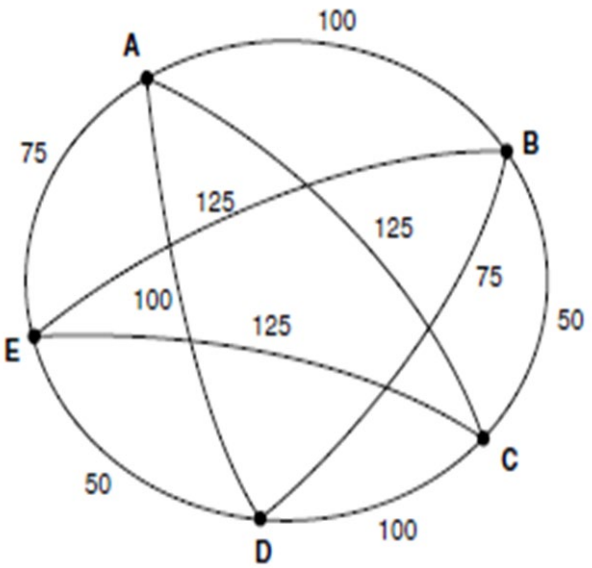
Търговски пътник

Да предположим, че търговец има пет града за посещение. Целта е да се намери най-късия маршрут, по който търговецът посещава всеки град точно един път и се връща в града, от който е тръгнал. Възлите на графа на пространството на състоянията са градовете, като на всяка дъга е присвоено тегло, указващо разходите за пътуване(разстояние).

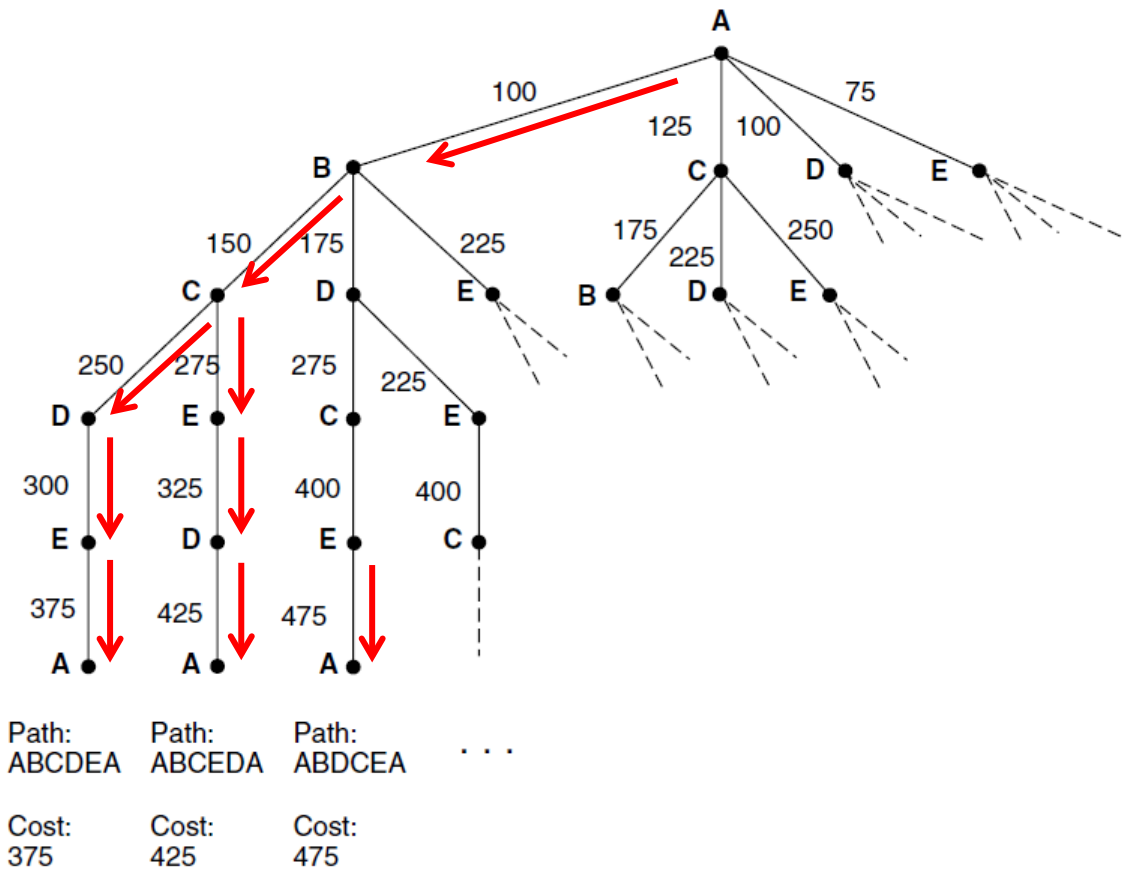
Ако началото е град А, възможните следващи състояния се добавят, докато всички градове се включат в маршрута. Целта е да се намери минималния път.

За 9 града може изчерпателно да се проверят всички пътища, но за 50 града, стратегията с пълно изчерпване не е изпълнима.

Пътят [A,D,C,B,E,A] с цена на пътя 450 км. е пример за решение на за задачата.



THE TRAVELING SALESPERSON
Search Strategy



Goal driven

Data driven

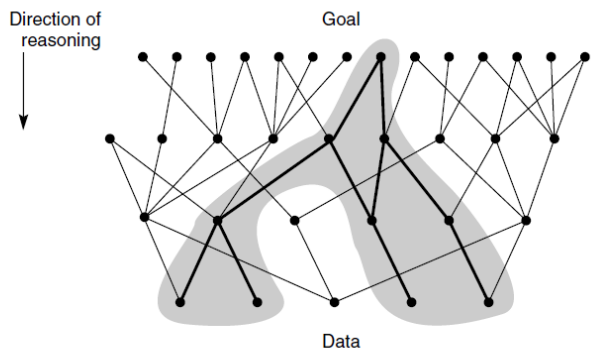


Figure 3.12 State space in which goal-directed search effectively prunes extraneous search paths.

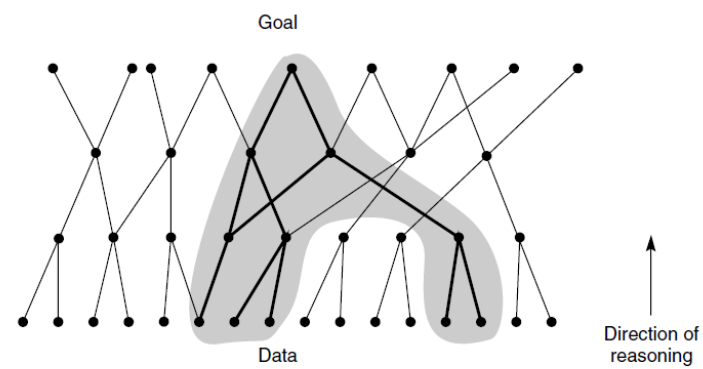


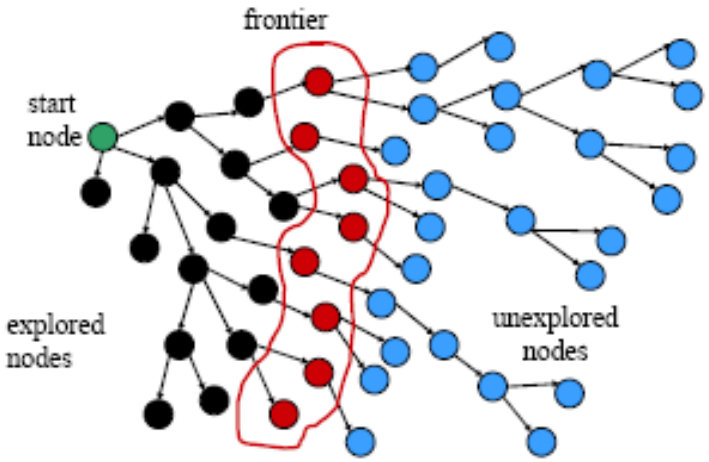
Figure 3.13 State space in which data-directed search prunes irrelevant data and their consequents and determines one of a number of possible goals.

Методи за неинформирано търсене

1. Breadth-first search (BFS): Обхождане в широчина
2. Depth-first search (DFS): Обхождане в дълбочина
3. Depth-limited search (DLS): Лимитирано обхождане в дълбочина.
4. Iterative-deepening search (IDS):Итеративно обхождане в дълбочина.
5. Uniform Cost Search(UCS): Търсене с равномерна цена на пътя

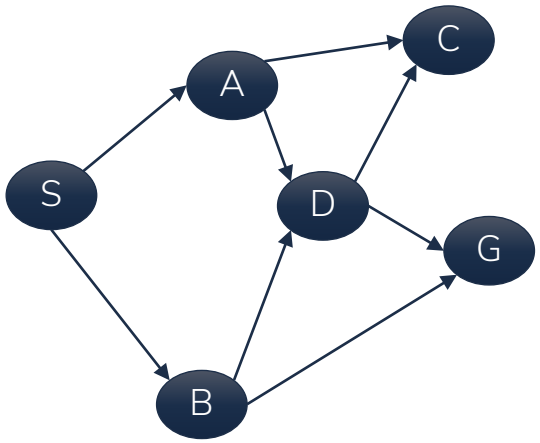
Пространство на състоянията стратегия за обхождане

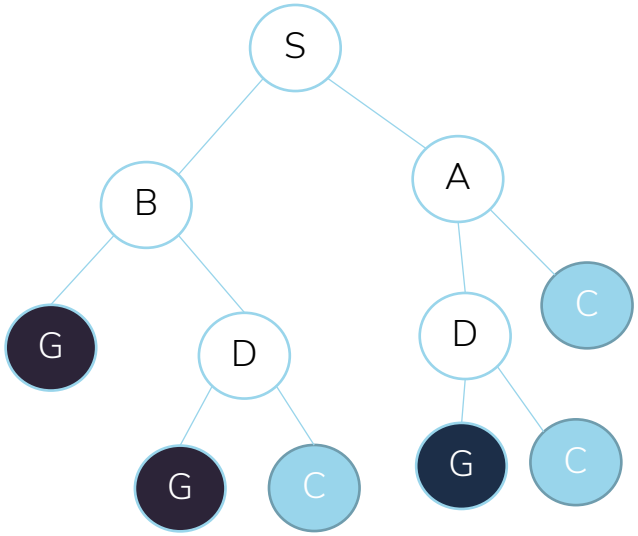
- *Проучени* (известен още като отворен списък с проучени възли) explored ,visited, closed
- *Фронт* (известен още като отворен списък, периферия) fringe, expanded, open
- *Непроучени*



Дърво на търсене

Пространство на състояния





Общ алгоритъм
за неинформирано търсене в дърво

```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

```
function EXPAND( node, problem) returns a set of nodes
    successors ← the empty set
    for each action, result in SUCCESSOR-FN[problem](STATE[node]) do
        s ← a new NODE
        PARENT-NODE[s] ← node;   ACTION[s] ← action;   STATE[s] ← result
        PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
        DEPTH[s] ← DEPTH[node] + 1
        add s to successors
    return successors
```

Общ алгоритъм
за неинформирано търсене в граф

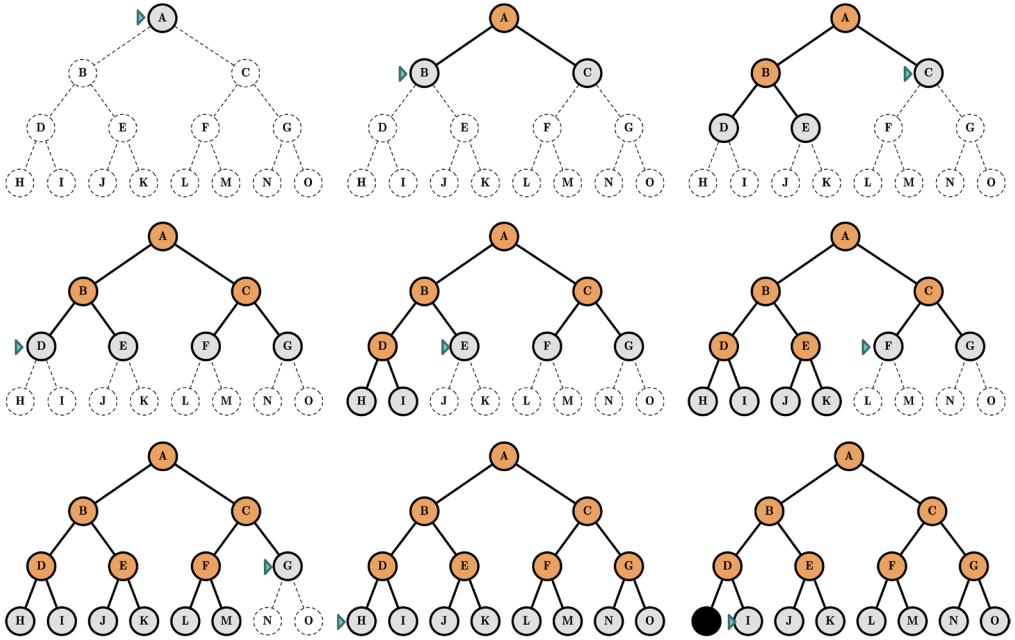
```
function GRAPH-SEARCH( problem, fringe) returns a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
        if STATE[node] is not in closed then
            add STATE[node] to closed
            fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

Обхождане в широчина
Breadth-First Search

BFS: Expand shallowest first.

Frontier

Explored



Artificial Intelligence. A Modern Approach. *Stuart Russell and Peter Norvig*

Алгоритъм за неинформирано търсене в широчина

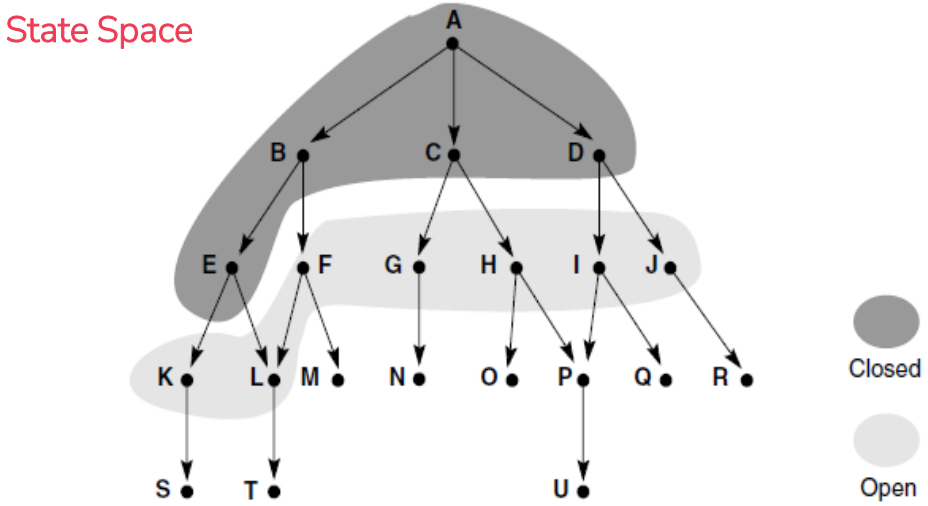
```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
     if EMPTY?(frontier) then return failure
     node ← POP(frontier)  /* chooses the shallowest node in frontier */
     add node.STATE to explored
     for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        if child.STATE is not in explored or frontier then
           if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
           frontier ← INSERT(child, frontier)
```

Figure 3.11 Breadth-first search on a graph.

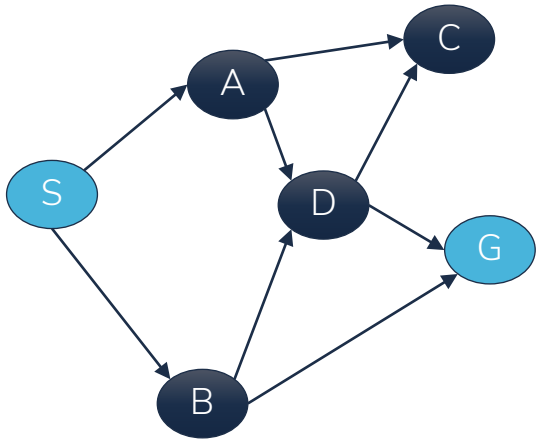
Artificial Intelligence. A Modern Approach. *Stuart Russell and Peter Norvig*

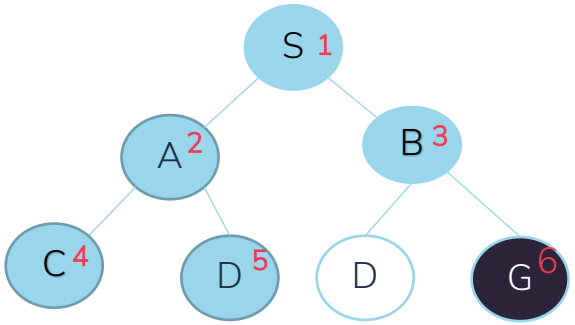
Пример 1 Breadth-First Search

1. open = [A]; closed = [ ]
2. open = [B,C,D]; closed = [A]
3. open = [C,D,E,F]; closed = [B,A]
4. open = [D,E,F,G,H]; closed = [C,B,A]
5. open = [E,F,G,H,I,J]; closed = [D,C,B,A]
6. open = [F,G,H,I,J,K,L]; closed = [E,D,C,B,A]
7. open = [G,H,I,J,K,L,M]
(as L is already on open); closed = [F,E,D,C,B,A]
8. open = [H,I,J,K,L,M,N]; closed = [G,F,E,D,C,B,A]
9. and so on until either U is found or open = [ ].



Пример
Обхождане в широчина





Open	Visited node
(S)	()
((S,A),(S,B))	(S)
((S,B),(S,A,C),(S,A,D))	(S,A)
((S,A,C),(S,A,D),(S,B,D),(S,B,G))	(S,B)
((S,A,D),(S,B,G),(S,D,C),(S,D,G))	(S,A,C)
((S,B,G))	(S,A,D)
	(S,B,G)

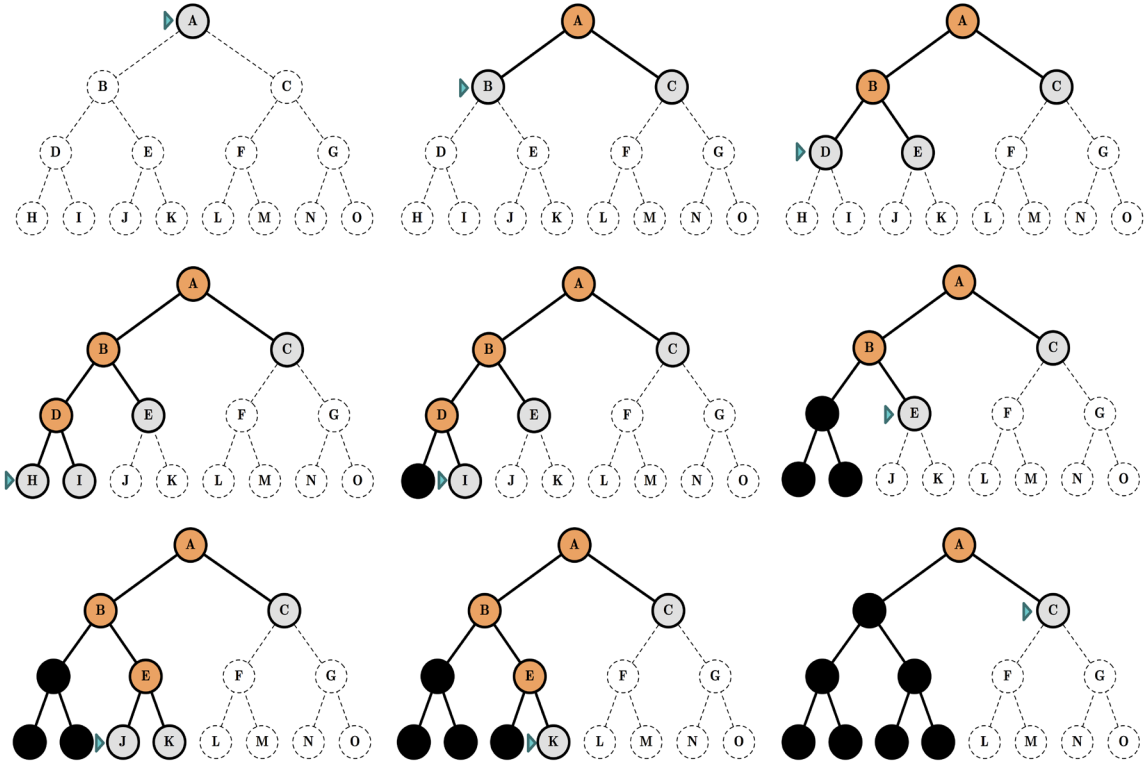
S->B->G

Обхождане в дълбочина
Depth-First Search

DFS: Expand deepest first in the current frontier of the search tree.

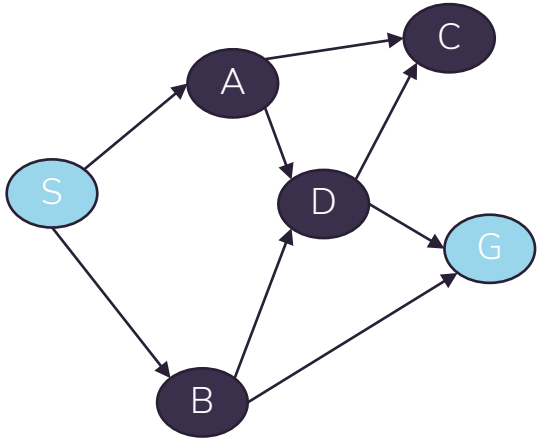
Frontier

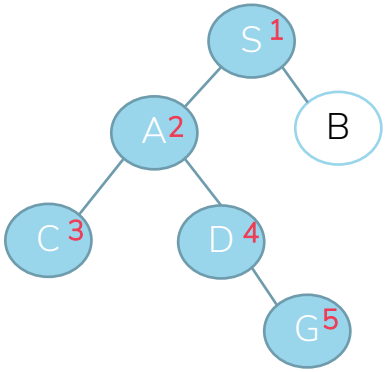
Explored



Artificial Intelligence. A Modern Approach. *Stuart Russell and Peter Norvig*

Пример
Обхождане в дълбочина



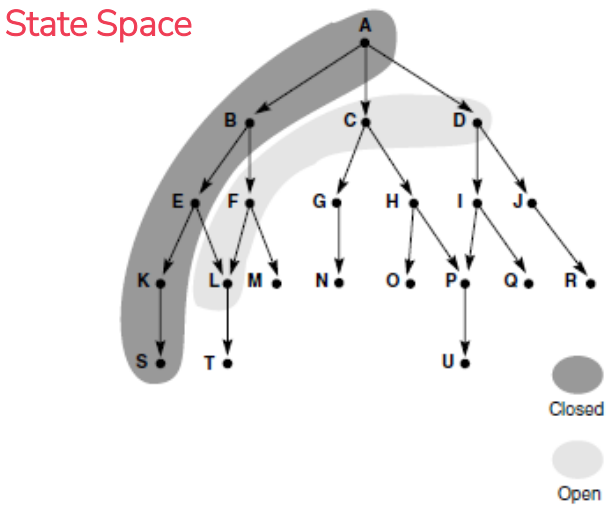


Open	Visited node
(S)	()
((S,B),(S,A))	(S)
((S,B),(S,A,D),(S,A,C))	(S,A)
((S,B),(S,A,D))	(S,A,C)
((S,B),(S,D,G),(S,D,C))	(S,A,D)
((S,B))	(S,D,G)

S->A->D->G

Обхождане в дълбочина
Depth-First Search A->U

1. open = [A]; closed = [ ]
2. open = [B,C,D]; closed = [A]
3. open = [E,F,C,D]; closed = [B,A]
4. open = [K,L,F,C,D]; closed = [E,B,A]
5. open = [S,L,F,C,D]; closed = [K,E,B,A]
6. open = [L,F,C,D]; closed = [S,K,E,B,A]
7. open = [T,F,C,D]; closed = [L,S,K,E,B,A]



Artificial intelligence Structures and Strategies for Complex Problem Solving
George F Luger

Обхождане в дълбочина

```
function DEPTH-FIRST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :

    frontier = Stack.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.pop()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.push(neighbor)

    return FAILURE
```

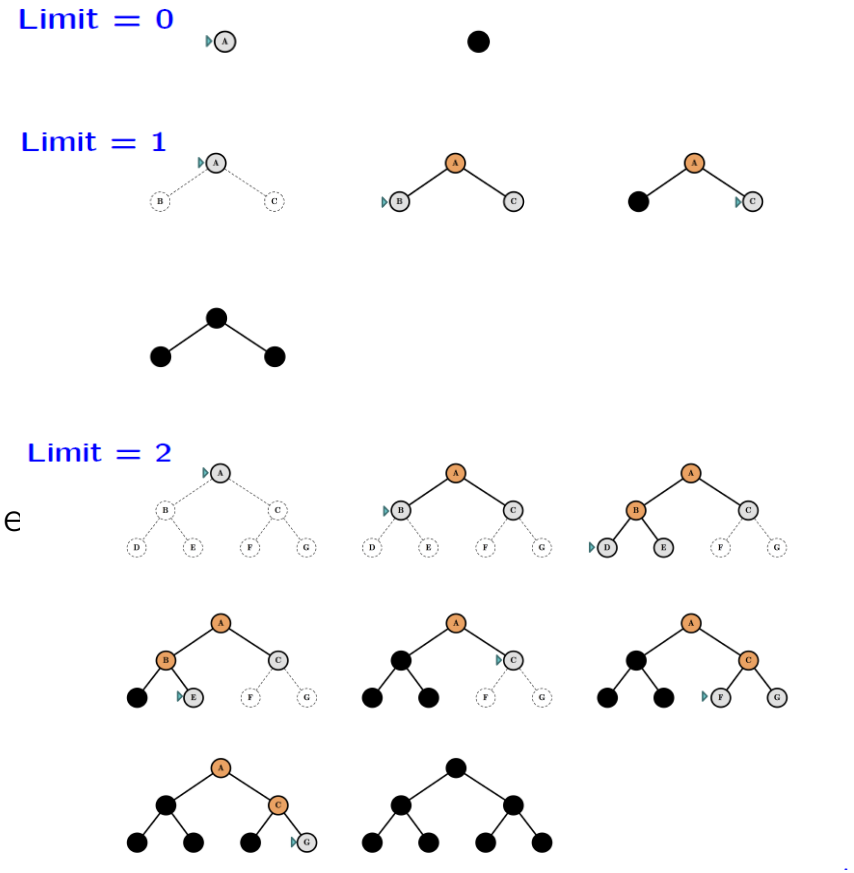
Рекурсивно обхождане в дълбочина

```
function depthsearch (current_state);                    % closed is global

begin
   if current_state is a goal
      then return SUCCESS;
   add current_state to closed;
   while current_state has unexamined children
      begin
         child := next unexamined child;
         if child not member of closed
            then if depthsearch(child) = SUCCESS
               then return SUCCESS
      end;
   return FAIL                                           % search exhausted
end
```

Depth Limited Iterative Deepening Search

- Depth Limited Search DLS- ограничава дълбочината на търсене
- Съчетава ползите от BFS и DFS.
- Лимитът на търсенето се увеличава итеративно до достигане на дълбочината на най-плиткото решение d.



Алгоритъм
за ограничено търсене в дълбочина

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns a solution, or failure/cutoff
  return RECURSIVE-DLS(MAKE-NODE(problem.INITIAL-STATE), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns a solution, or failure/cutoff
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  else if limit = 0 then return cutoff
  else
      cutoff_occurred? ← false
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          result ← RECURSIVE-DLS(child, problem, limit − 1)
          if result = cutoff then cutoff_occurred? ← true
          else if result ≠ failure then return result
      if cutoff_occurred? then return cutoff else return failure
```

Iterative Deepening Search

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
    for depth = 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH(problem, depth)
        if result ≠ cutoff then return result
```

Figure 3.18 The iterative deepening search algorithm, which repeatedly applies depth-limited search with increasing limits. It terminates when a solution is found or if the depth-limited search returns *failure*, meaning that no solution exists.

Iterative Deepening Search

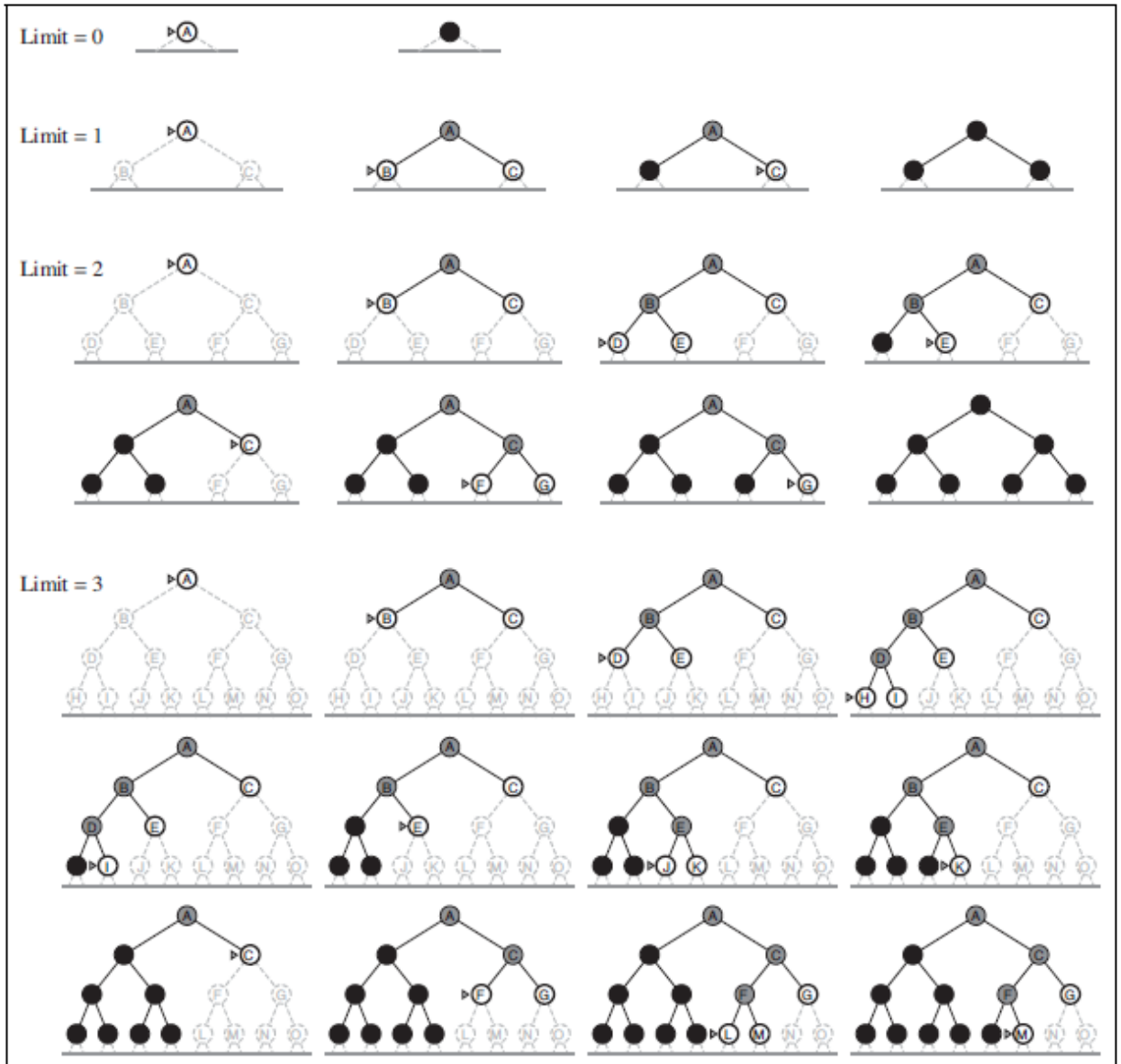
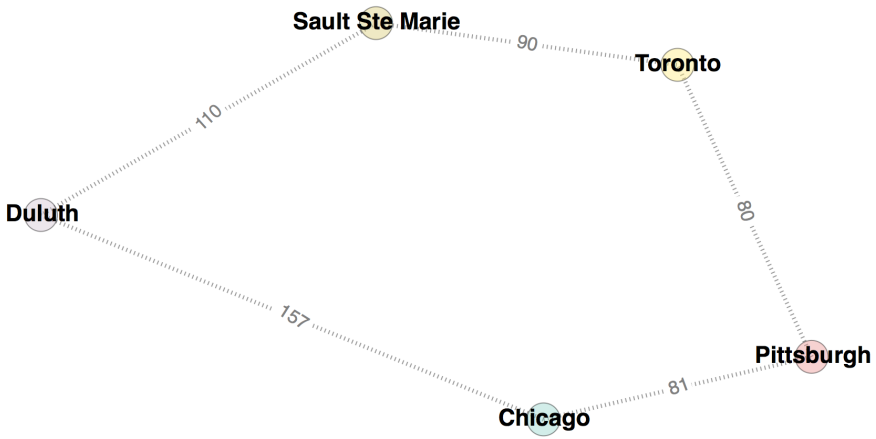


Figure 3.19 Four iterations of iterative deepening search on a binary tree.

Неинформирано търсене с равномерна цена на пътя (Uniform-cost search)

- При идентична цена на стъпката търсенето в широчина е оптимално, защото стратегията винаги разширява най-плиткия неразгънат възел.
- Стратегията с равномерна цена на пътя разширява възела с *най-малката цена на пътя (cost* g(n)).
- Реализацията на фронта е с приоритетна опашка, сортирана във възходящ ред по цената на пътя .
- BFS е частен случай на търсене с равномерна цена на пътя

Go from Chicago to Sault Ste Marie.



Using BFS, we would end Chicago-Duluth-Sault Ste Marie, 267. However, using UCS, we would end Chicago-Pittsburgh-Toronto-Sault Ste Marie 251, which is actually the shortest path!

Uniform-cost search

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
     if EMPTY?(frontier) then return failure
     node ← POP(frontier)  /* chooses the lowest-cost node in frontier */
     if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
     add node.STATE to explored
     for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        if child.STATE is not in explored or frontier then
           frontier ← INSERT(child, frontier)
        else if child.STATE is in frontier with higher PATH-COST then
           replace that frontier node with child
```

Figure 3.14 Uniform-cost search on a graph. The algorithm is identical to the general graph search algorithm in Figure 3.7, except for the use of a priority queue and the addition of an extra check in case a shorter path to a frontier state is discovered. The data structure for *frontier* needs to support efficient membership testing, so it should combine the capabilities of a priority queue and a hash table.

Румъния

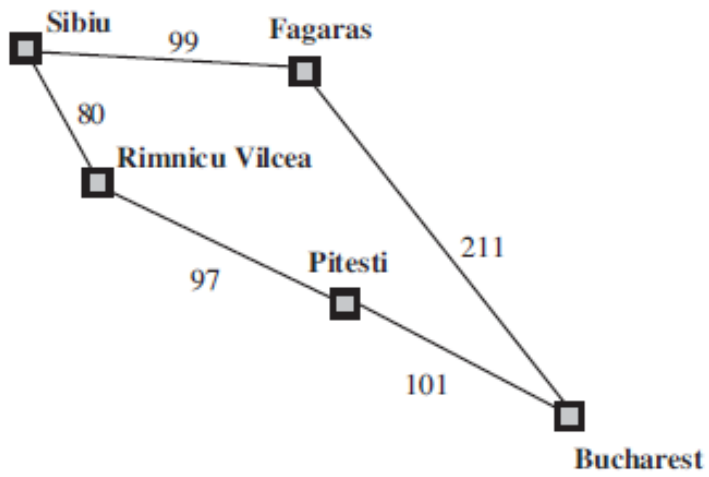
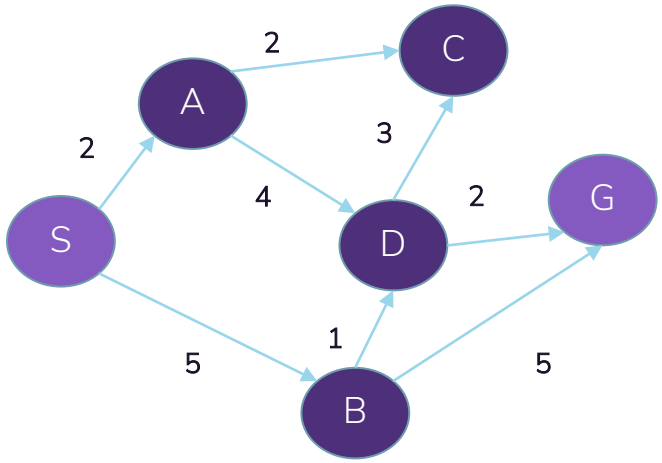
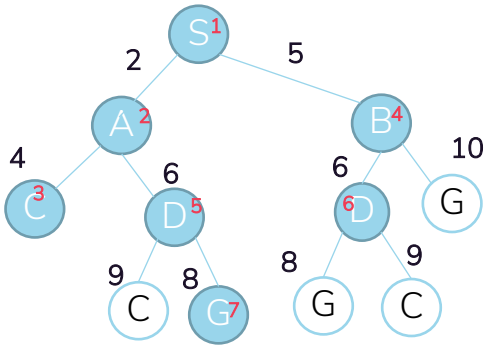


Figure 3.15 Part of the Romania state space, selected to illustrate uniform-cost search.

Uniform cost search

Пример





Open	Visited node
(S)	(S)
((S,A,2),(S,B,5))	(S,A,2)
((S,A,C,4), (S,B,5),(S,A,D,6))	(S,A,C,4)
((S,B,5),(S,A,D,6))	(S,B,5)
((S,A,D,6),(S,B,G,10))	(S,A,D,6)
((S,A,D,G,8))	**(S,A,D,G,8)**
()	
S->A->D->G	

BFS Оценка

Пълнота: Пълна стратегия

Време: $1+b+b^2 + b^3 +\ldots+ b^d = O(b^d)$

Памет: $O(b^d)$

Оптималност : Yes (if cost = 1 per step).

Имплементация: Фронт: FIFO (Queue)

Note: Ако тестът за достигане на целта се прилага при разширяване, а не при генериране на наследници $O(b^{d+1})$

DFS оценка

Пълнота: не е пълен

(завършва и стратегията е пълна, когато графът на състоянията е краен).

Време: O(b^m): 1+ b +b^2 + b^3 +...+ b^m = O(b^m)

(ако m е много по-голям от d, но ако решенията са компактни, може да е много по-бърза от BFS.)

Памет: O(bm) linear space complexity!

(съхранява се само един път от корена към листата заедно с останалите неразширени съседни възли.

Оптималност: не е оптимална

Implementation: Фронт: LIFO (Stack)

Резюме на алгоритмите за неинформирано търсене

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes^a	Yes^{a,b}	No	No	Yes^a	Yes^{a,d}
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes^c	Yes	No	No	Yes^c	Yes^{c,d}

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: ^{a} complete if b is finite; ^{b} complete if step costs $\geq \epsilon$ for positive ϵ; ^{c} optimal if step costs are all identical; ^{d} if both directions use breadth-first search.

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

Figure 3.13 Time and memory requirements for breadth-first search. The numbers shown assume branching factor $b = 10$; 1 million nodes/second; 1000 bytes/node.

Artificial Intelligence. A Modern Approach. *Stuart Russell and Peter Norvig*

Резюме на алгоритмите за неинформирано търсене

Методи, които не използват специфична информация за областта на проблема.

Breadth-first search разширява първо най-плитките възли; тя е пълна, оптимална за равна цена на стъпката, но има експоненциална сложност.

Uniform-cost search разширява възела с минимални разходи за път, g(n), и е оптимална.

Depth-first search първо разширява най-дълбокият неразгънат възел. Тя не е нито пълна, нито оптимална, но има линейна пространствена сложност.

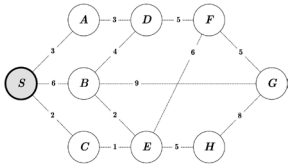
Depth-limited search добавя лимит на дълбочината.

Iterative deepening search извиква depth-first search с нарастващ лимит на дълбочина, докато се намери целта. Тя е пълна, оптимална (равна цена на стъпка), има времева сложност сравнима с breadth-first search и линейна пространствена сложност.

Задачи

B F S

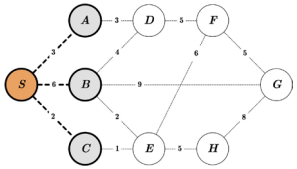
Start **S** , Goal **G**



Queue:

S

Order of Visit:

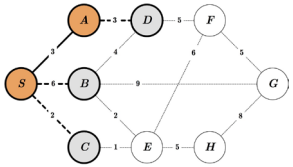


Queue:

S	A	B	C

Order of Visit:

S

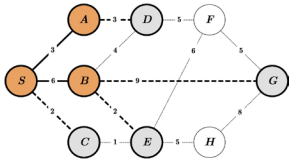


Queue:

S	A	B	C	D

Order of Visit:

S A

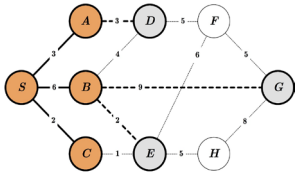


Queue:

S	A	B	C	D	E	G

Order of Visit:

S A B

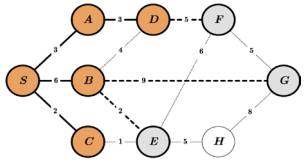


Queue:

S	A	B	C	D	E	G

Order of Visit:

S A B C

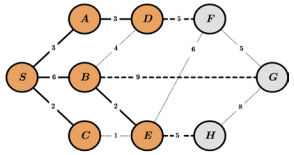


Queue:

S	A	B	C	D	E	G	F

Order of Visit:

S A B C D

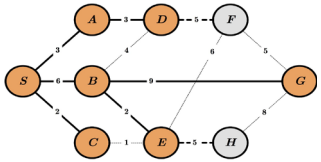


Queue:

S	A	B	C	D	E	G	F	H

Order of Visit:

S A B C D E

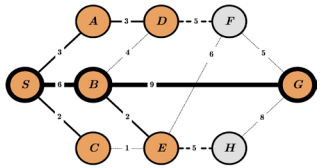


Queue:

S	A	B	C	D	E	G	F	H

Order of Visit:

S A B C D E G



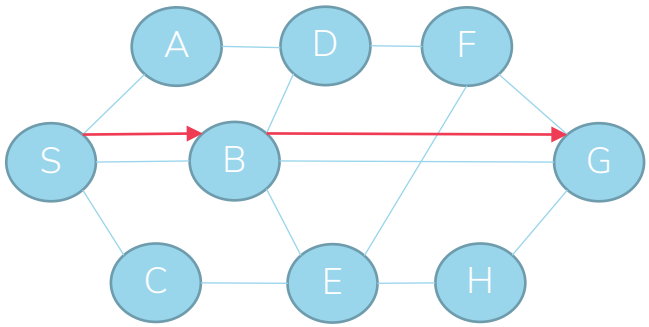
Queue:

S	A	B	C	D	E	G	F	H

Order of Visit:

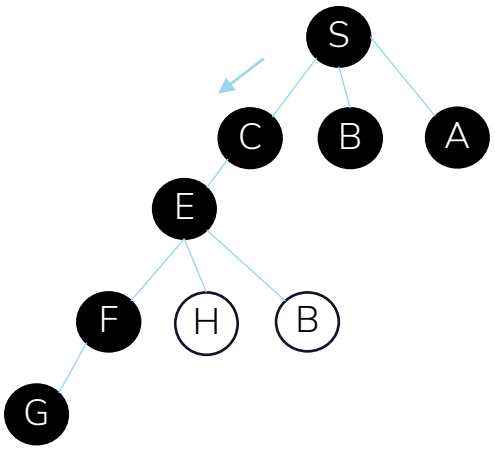
S A B C D E G

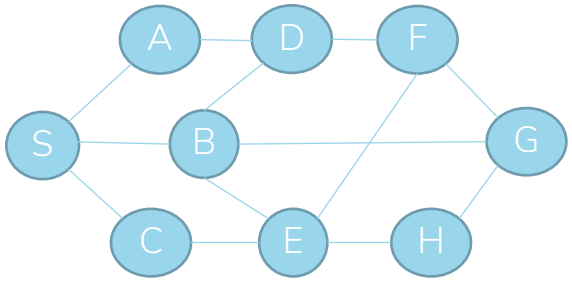
BFS



Open	Visited node
(S)	()
(A,B,C)	S
(B,C,D)	A
(C,D,G)	B
(G,E)	C
(G)	*G*

DFS

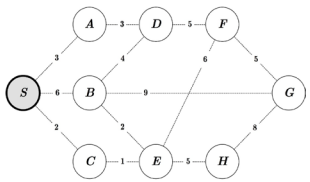




Open	Visited
(S)	()
(A,B,C)	S
(A,B,E)	(S,C)
(A,B,F,H)	(C,E)
(A,B,F,G)	(E,H)
(A,B,F)	*(G)*

Open	Visited
(S)	()
(C,B,A)	S
(C,B,D)	(S,A)
(A,B,F)	(A,D)
(A,B,G)	(D,F)
(A,B)	*(G)*

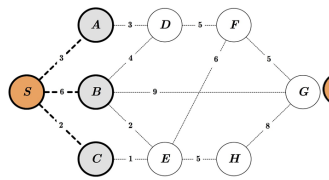
D F S



Stack:

S

Order of Visit:

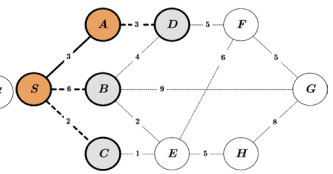


Stack:

S	*C*	*B*	*A*

Order of Visit:

S

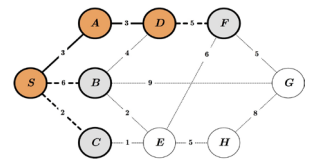


Stack:

S	*C*	*B*	*A*	*D*

Order of Visit:

S *A*

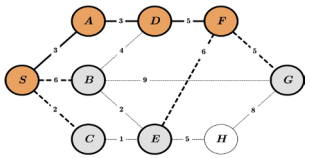


Stack:

S	*C*	*B*	*A*	*D*	*F*

Order of Visit:

S *A* *D*

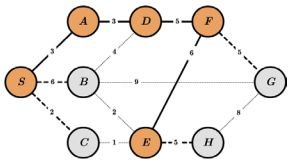


Stack:

S	*C*	*B*	*A*	*D*	*F*	*G*	*E*

Order of Visit:

S *A* *D* *F*

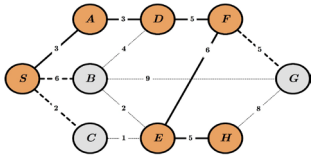


Stack:

S	*C*	*B*	*A*	*D*	*F*	*G*	*E*	*H*

Order of Visit:

S *A* *D* *F* *E*

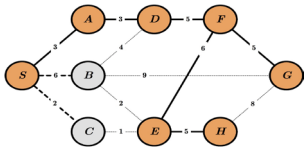


Stack:

S	*C*	*B*	*A*	*D*	*F*	*G*	*E*	*H*

Order of Visit:

S *A* *D* *F* *E* *H*

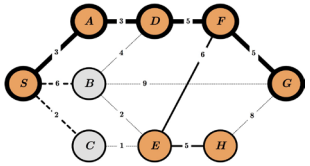


Stack:

S	*C*	*B*	*A*	*D*	*F*	*G*	*E*	*H*

Order of Visit:

S *A* *D* *F* *E* *H* *G*



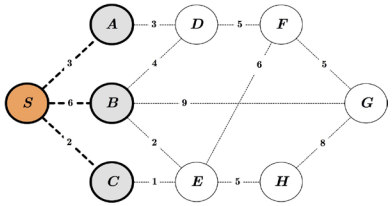
Stack:

S	*C*	*B*	*A*	*D*	*F*	*G*	*E*	*H*

Order of Visit:

S *A* *D* *F* *E* *H* *G*

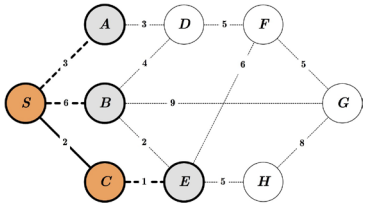
UCS



Priority Queue:

S_0	C_2	A_3	B_6

Order of Visit:

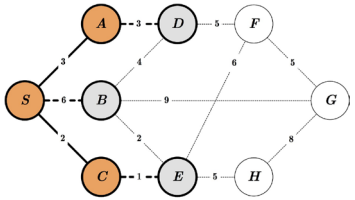


Priority Queue:

S_0	C_2	A_3	E_3	B_6

Order of Visit:

S

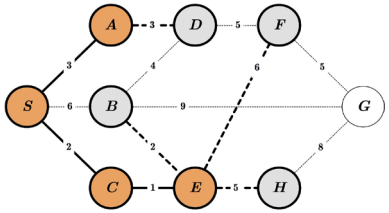


Priority Queue:

S_0	C_2	A_3	E_3	B_6	D_6

Order of Visit:

S C

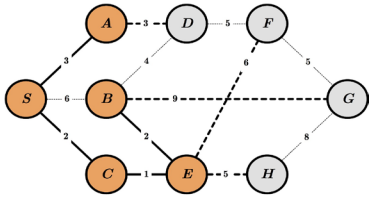


Priority Queue:

S_0	C_2	A_3	E_3	B_5	D_6	H_8	F_9

Order of Visit:

S C A

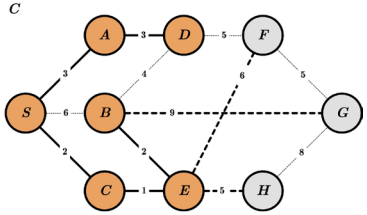


Priority Queue:

S_0	C_2	A_3	E_3	B_5	D_6	H_8	F_9	G_{14}

Order of Visit:

S C A E

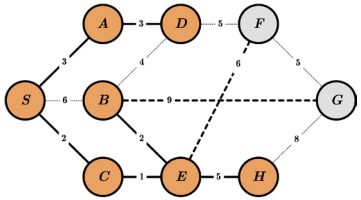


Priority Queue:

S_0	C_2	A_3	E_3	B_5	D_6	H_8	F_9	G_{14}

Order of Visit:

S C A E B

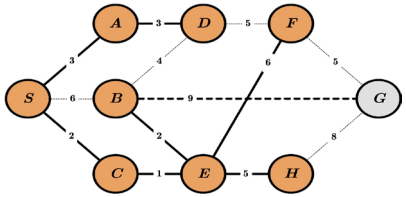


Priority Queue:

S_0	C_2	A_3	E_3	B_5	D_6	H_8	F_9	G_{14}

Order of Visit:

S C A E B D

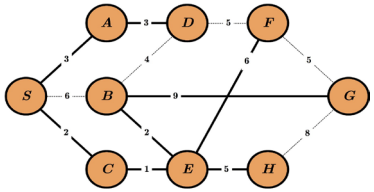


Priority Queue:

S_0	C_2	A_3	E_3	B_5	D_6	H_8	F_9	G_{14}

Order of Visit:

S C A E B D H

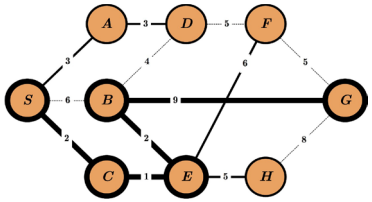


Priority Queue:

S_0	C_2	A_3	E_3	B_5	D_6	H_8	F_9	G_{14}

Order of Visit:

S C A E B D H F



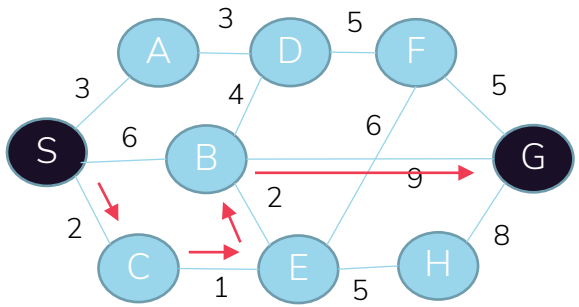
Priority Queue:

S_0	C_2	A_3	E_3	B_5	D_6	H_8	F_9	G_{14}

Order of Visit:

S C A E B D H F G

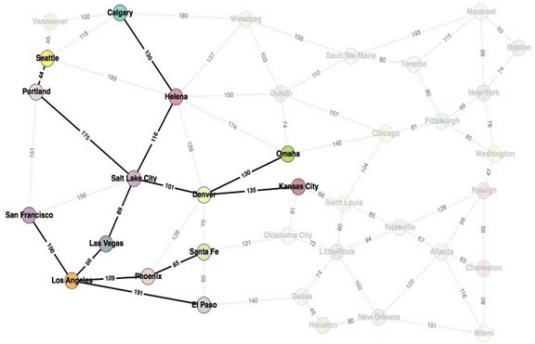
Uniform cost search

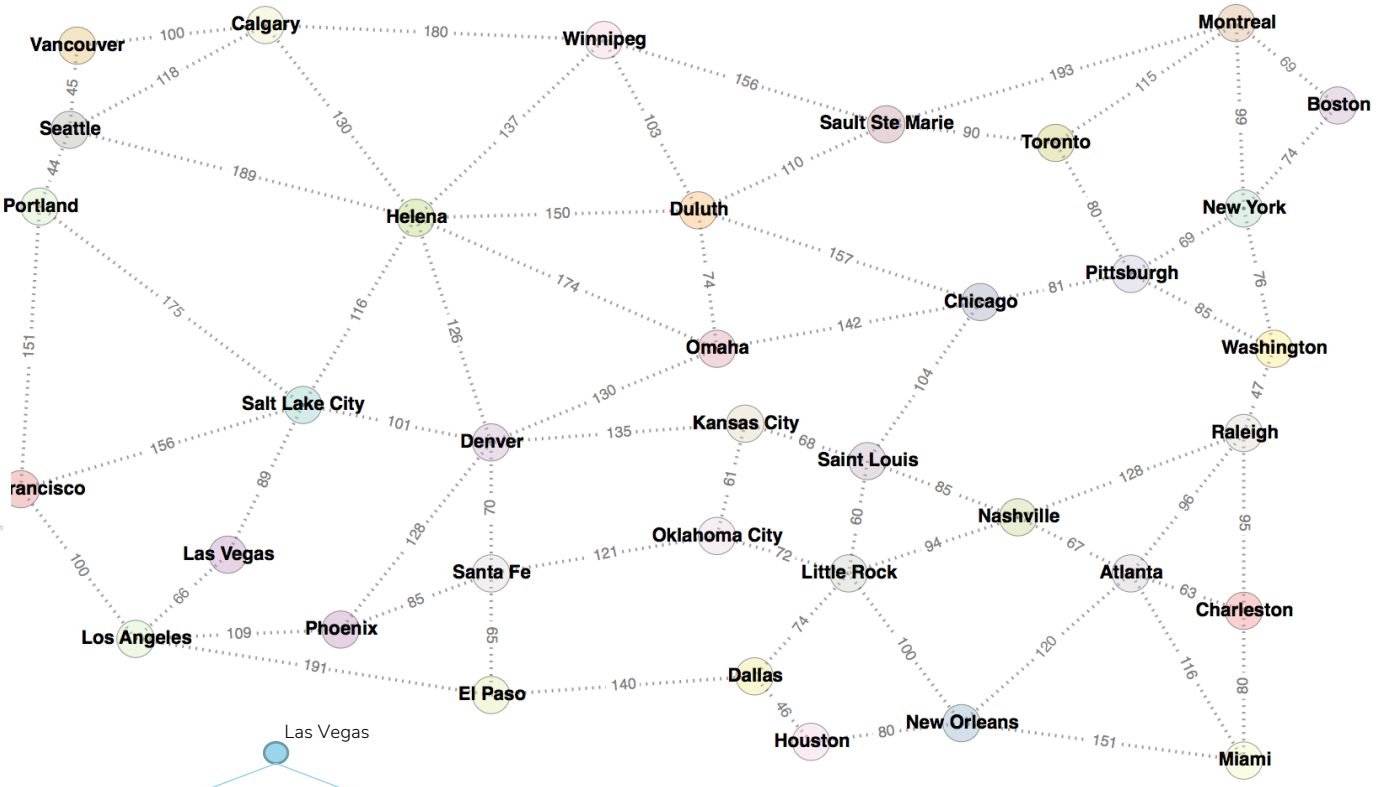


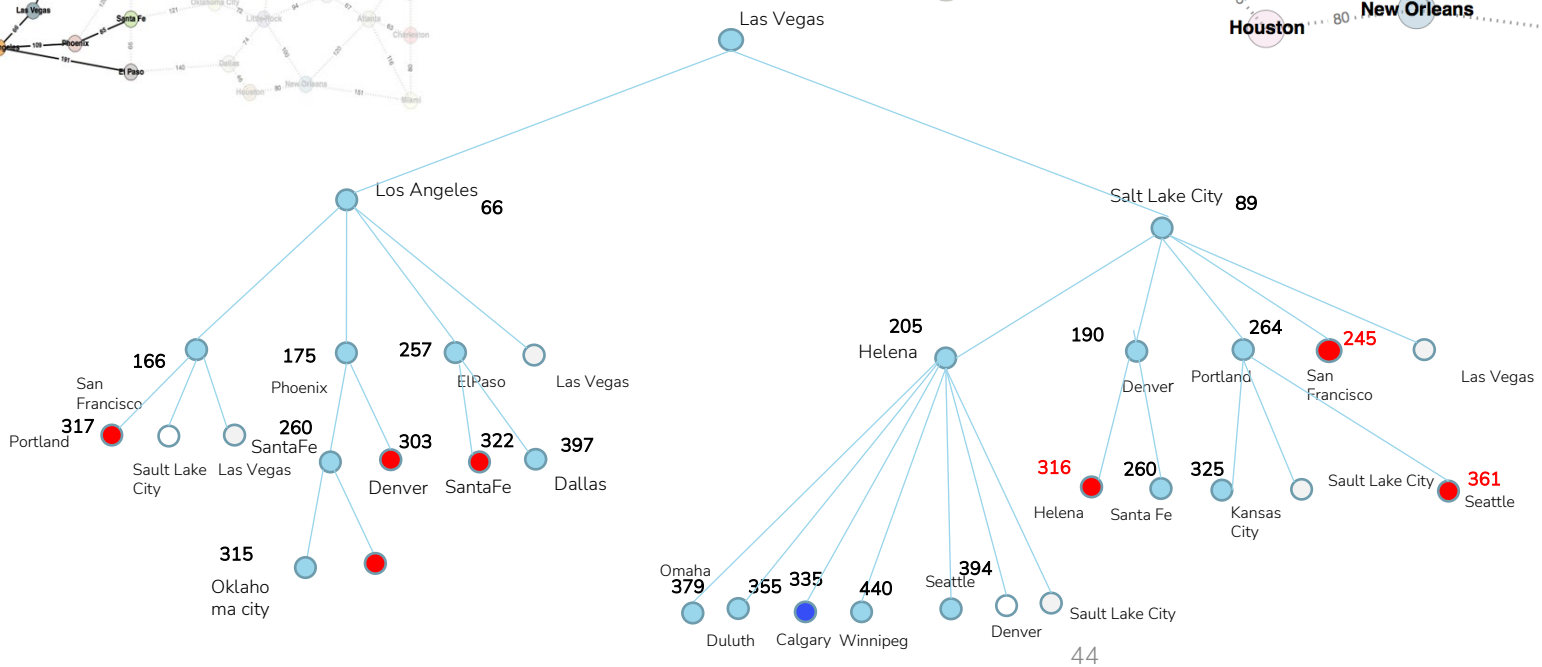
Open	Visited node.
(S)	()
((S,C,2), (S,A,3),(S,B,6))	(S)
((S,A,3), (S,C,E,3),(S,B,6))	((S),(C,2))
((S,C,E,3),(S,B,6),(S,A,D,6))	((S),(C,2),(A,3))
((S,E,B,5),(S,A,D,6),(S,C,E,H,8),(S,C,E,F,9))	((S),(C,2),(A,3),(E,3))
((S,A,D,6),(S,C,E,H,8),(S,C,E,F,9),(S,B,G,14))	((S),(C,2),(A,3),(E,3),(B,5))
((S,C,E,H,8),(S,C,E,F,9),(S,A,D,F,11),(S,B,G,14),(S,A,F,G,16))	((S),(C,2),(A,3),(E,3),(B,5),(D,6))
((E,F,9),(B,G,14),(S,C,E,H,G,16),(S,A,D,F,G,16))	((S),(C,2),(A,3),(E,3),(B,5),(D,6),(H,8))
((S,C,E,B,G,14),(S,C,E,F,G,14))	((S),(C,2),(A,3),(E,3),(B,5),(D,6),(H,8),(F,9))
()	((S),(C,2),(A,3),(E,3),(B,5),(D,6),(H,8),(F,9),(G,14))

UCS
Start: Las Vegas
Goal: Calgary

Order of Visit: Las Vegas, Los Angeles, Salt Lake City, San Francisco, Phoenix, Denver, Helena, El Paso, Santa Fe, Portland, Seattle, Omaha, Kansas City, Calgary

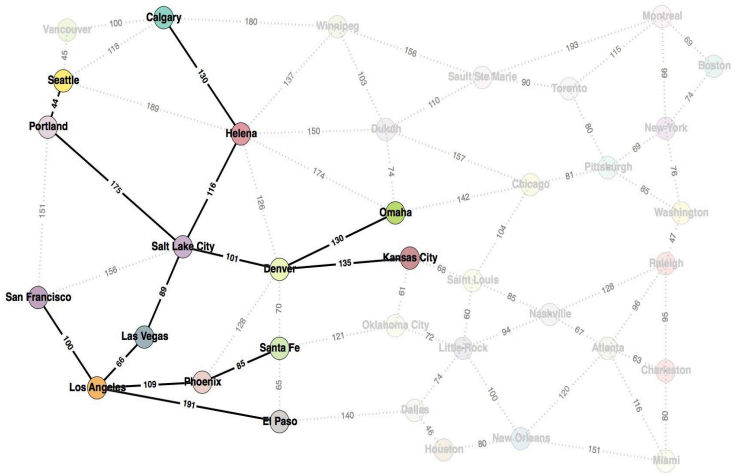


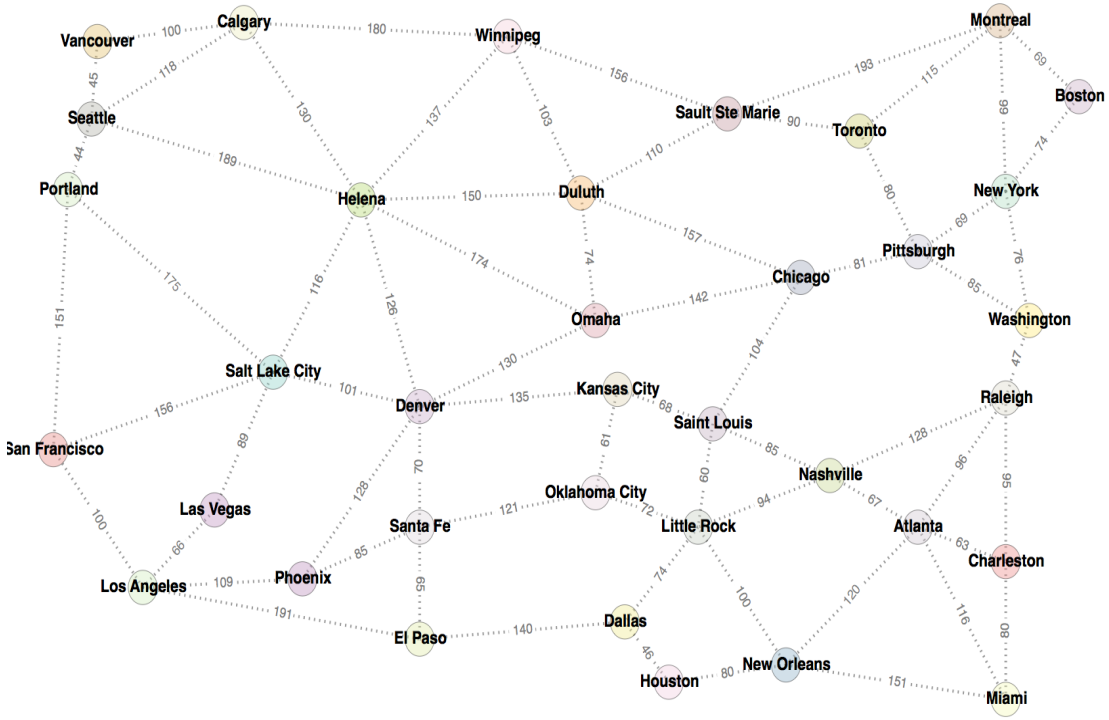




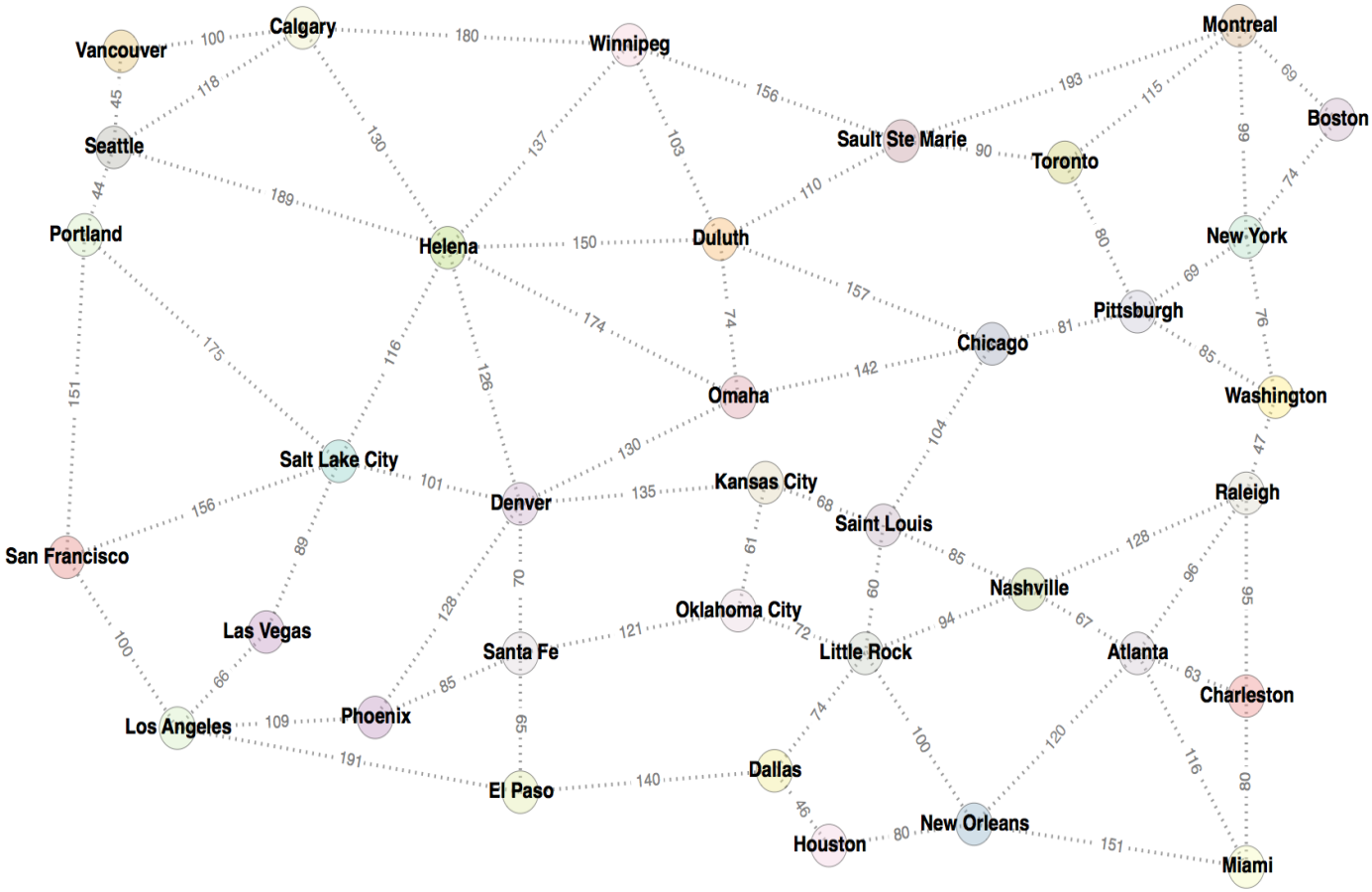
UCS
Start: Las Vegas
Goal: Calgary

Order of Visit: Las Vegas, Los Angeles, Salt Lake City, San Francisco, Phoenix, Denver, Helena, El Paso, Santa Fe, Portland, Seattle, Omaha, Kansas City, Calgary



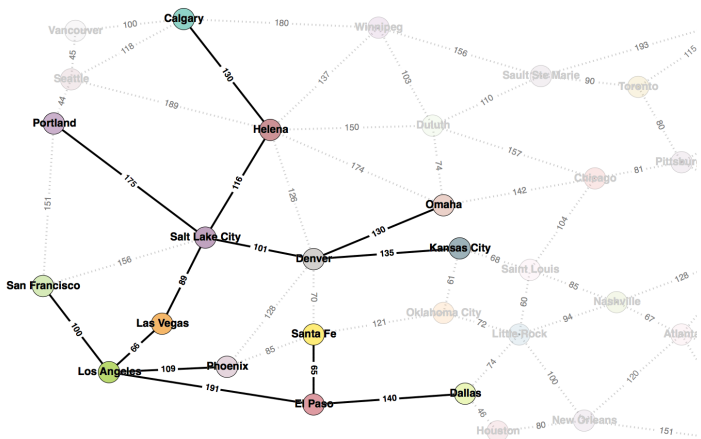


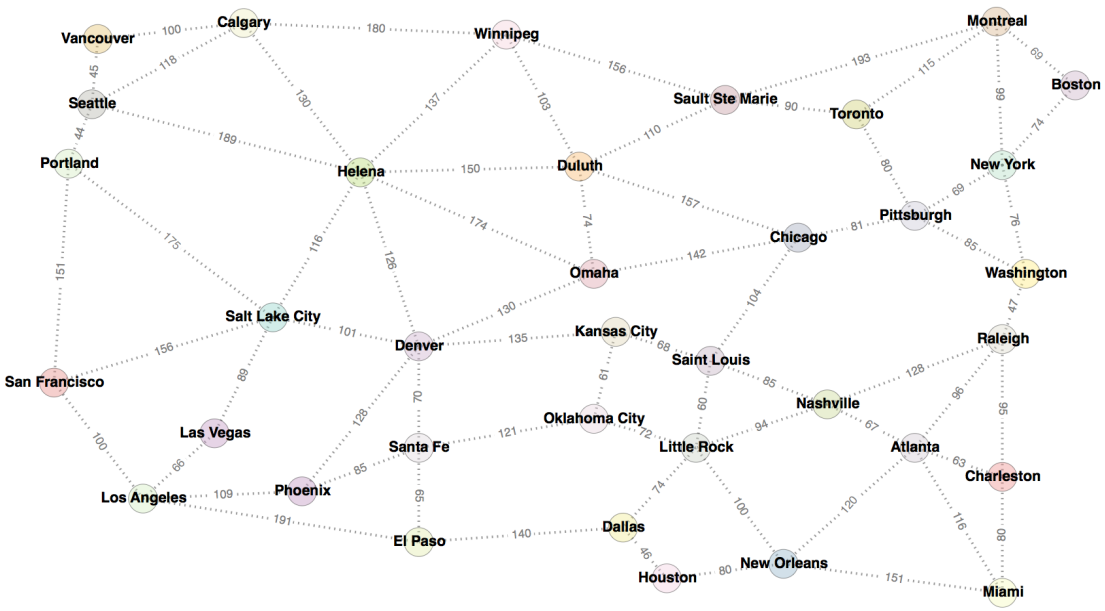
DFS
Start: La
Goal: Ca



BFS
Start: Las Vegas
Goal: Calgary

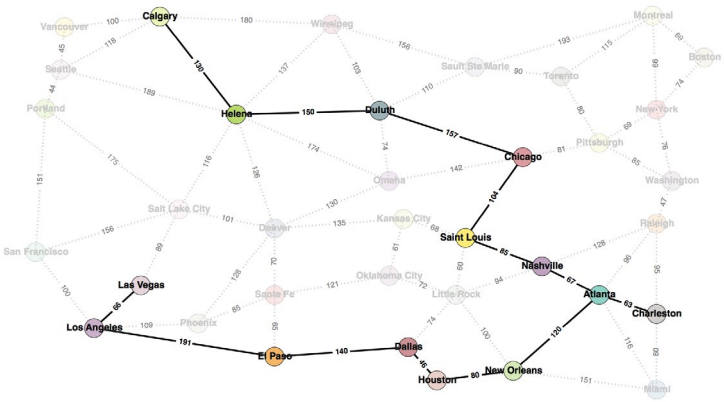
Order of Visit: Las Vegas, Los Angeles, Salt Lake City, El Paso, Phoenix, San Francisco, Denver, Helena, Portland, Dallas, Santa Fe, Kansas City, Omaha, Calgary.

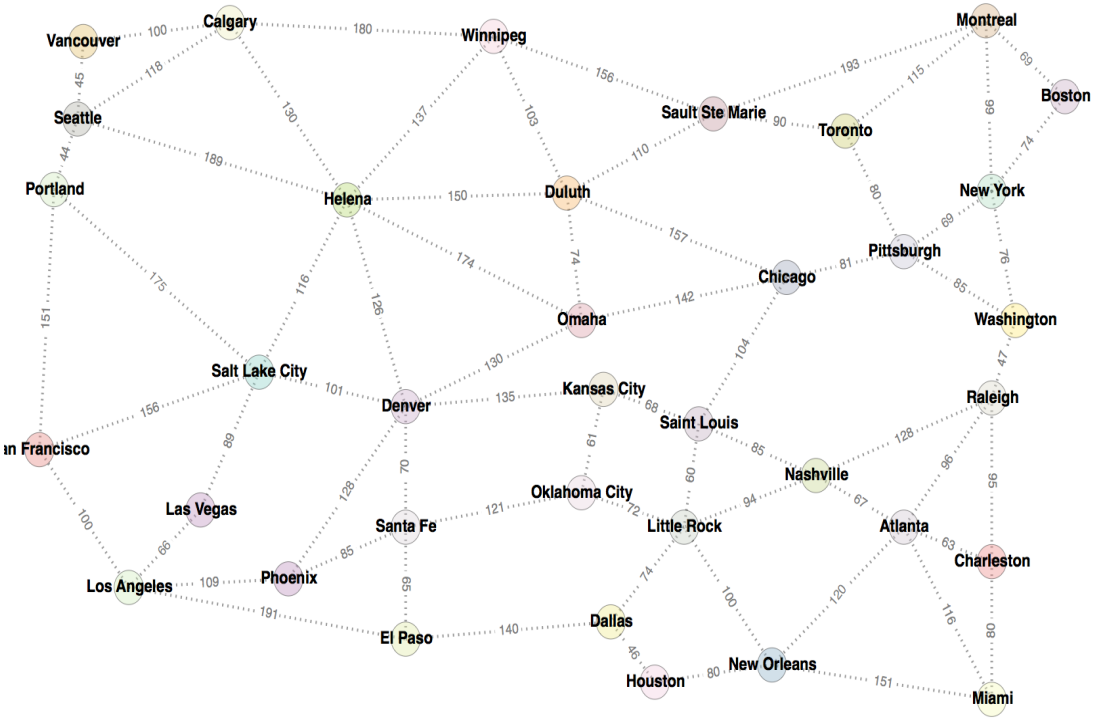




DFS
Start: Las Vegas
Goal: Calgary

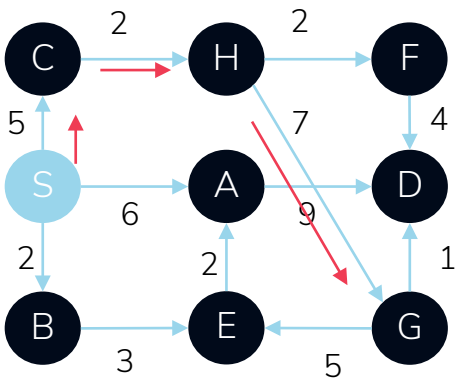
Order of Visit: Las Vegas, Los Angeles, El Paso, Dallas, Houston, New Orleans, Atlanta, Charleston, Nashville, Saint Louis, Chicago, Duluth, Helena, Calgary.





DFS

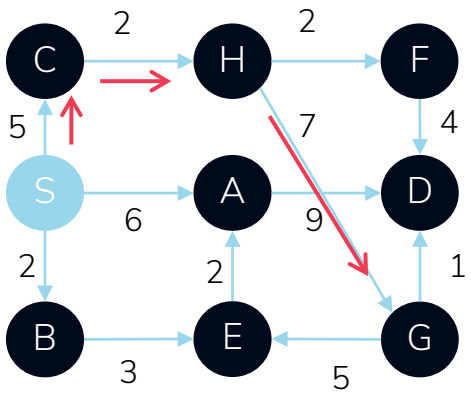
S->G



Open	Visited
(S)	()
((S,A),(S,B),**(S,C)**)	(S)
((S,A),(S,B),**(S,C,H)**)	(S,C)
((S,A),(S,B),(S,C,H,F),**(S,C,H,G)**)	(S,C,H)
((S,A),(S,B),(S,C,H,F))	**(S,C,H,G)**

BFS

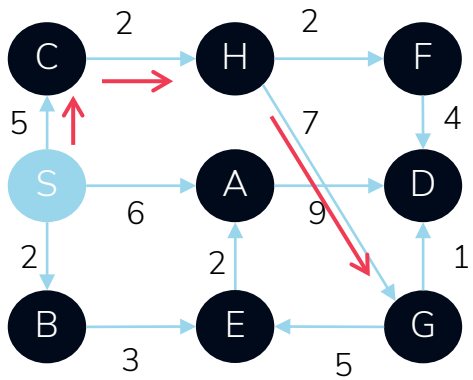
S->G



Open	Visited
(S)	()
((S,A),(S,B),(S,C))	(S)
((S,B),(S,C),(S,A,D))	(S,A)
((S,C),(S,A,D),(S,B,E))	(S,A,B)
((S,A,D),(S,B,E),(S,C,H))	(S,A,B,C)
((S,B,E),(S,C,H))	(S,A,B,C,D)
((S,C,H),(S,E,A))	(S,A,B,C,D,E)
((S,C,H,F),(S,C,H,G))	(S,A,B,C,D,E,H)
((S,CH,G))	(S,A,B,C,D,E,G)

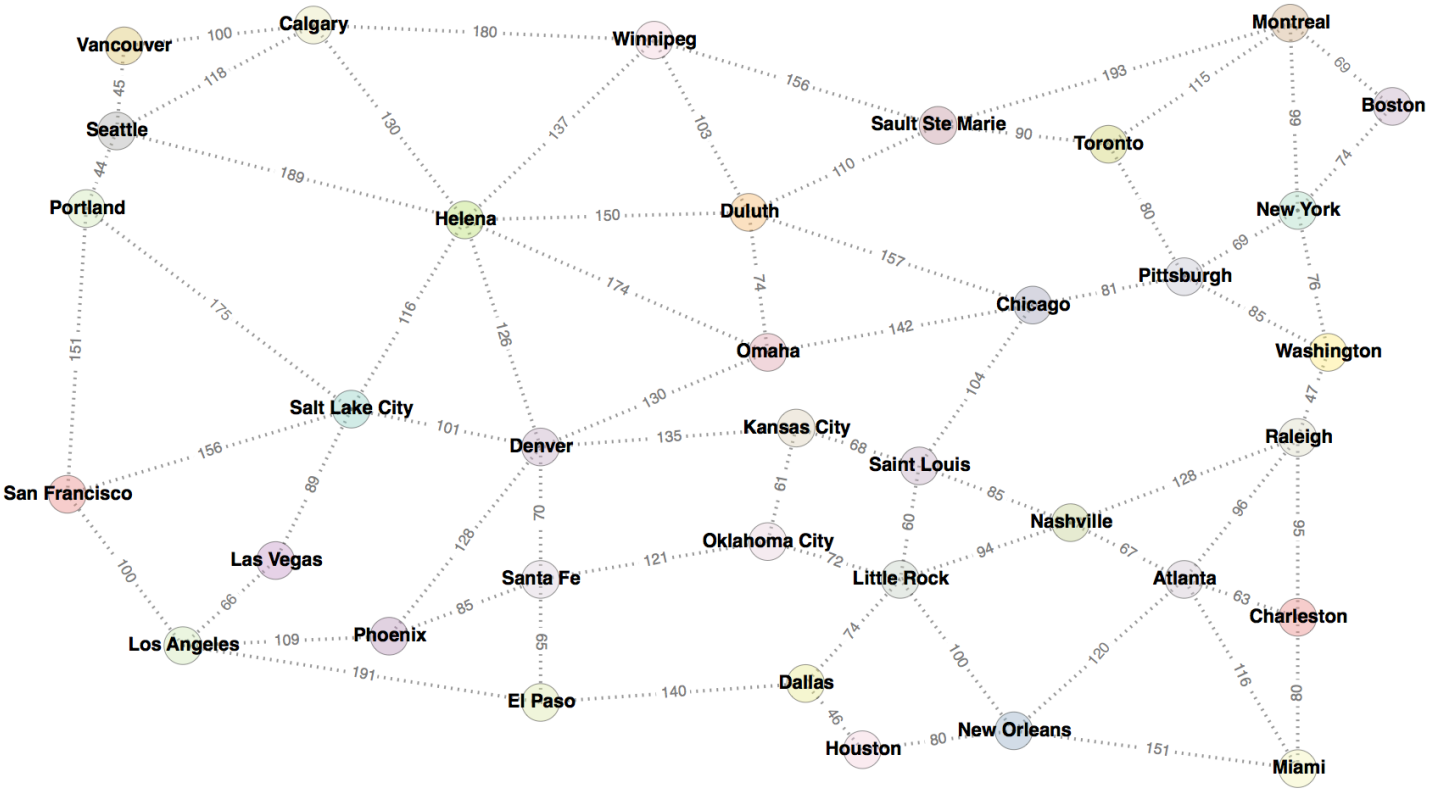
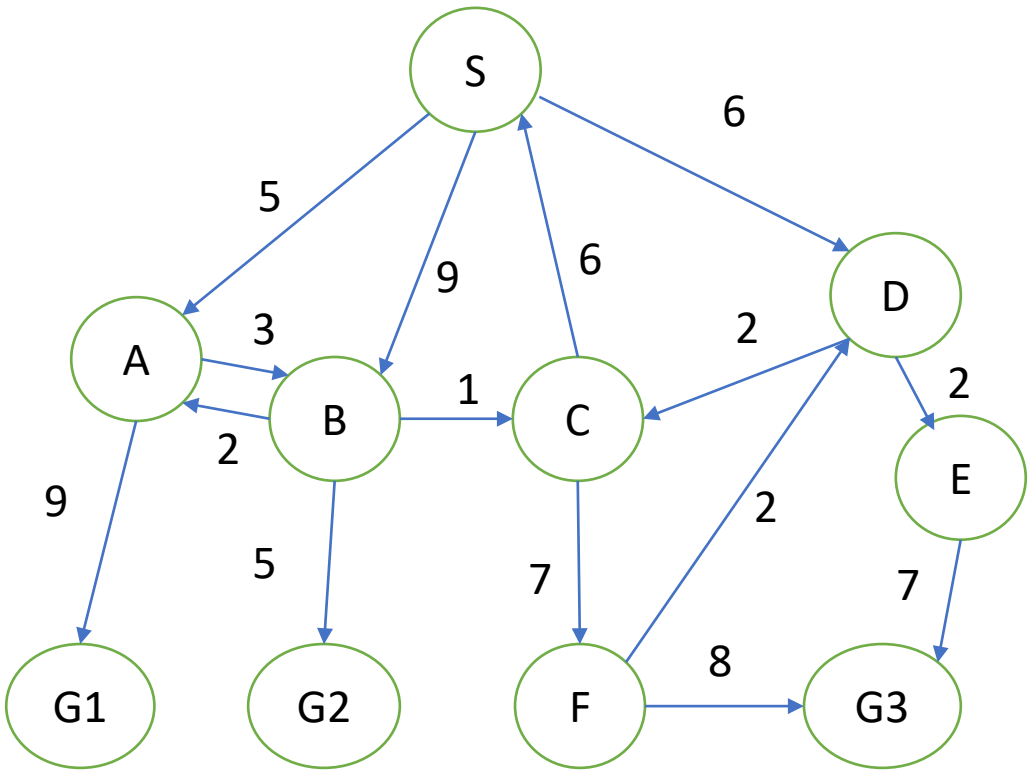
UCS

S->G



Open	Visited
(S)	()
((S,B,2),(S,C,5), (S,A,6))	(S)
(**(S,C,5)**,(S,A,6),(B,E,5))	((S),(B,2))
(**(S,A,6)**,(S,B,E,5),(S,C,H,7))	((S),(B,2),(C,5))
(**(S,B,E,5)**,(S,C,H,7),(S,A,D,15))	((S),(B,2),(C,5),(A,6))
(**(S,C,H,7)**,(S,B,E,A,7))	((S),(B,2),(C,5),(A,6),(E,5))
(**(S,C,H,F,9),**(S,C,H,G,14))	((S),(B,2),(C,5),(A,6),(E,5),(H,7))
(S,C,H,F,D,3),(S,C,H,G,14))	((S),(B,2),(C,5),(A,6),(E,5),(H,7),(F,9))
((F,D,13),(H,G,14))	((S),(B,2),(C,5),(A,6),(E,5),(H,7),(F,9),(D,13))
()	((S),(B,2),(C,5),(A,6),(E,5),(H,7),(F,9),(D,13) (G,14))

Chicago->Helena



UCS