Севастопольский государственный университет
Институт информационных технологий

“МЕТОДЫ И СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА" (МиСИИ)

Бондарев Владимир Николаевич

Лекция 4

МЕТОДЫ НЕИНФОРМИРОВАННОГО ПОИСКА РЕШЕНИЙ ЗАДАЧ В ПРОСТРАНСТВЕ СОСТОЯНИЙ

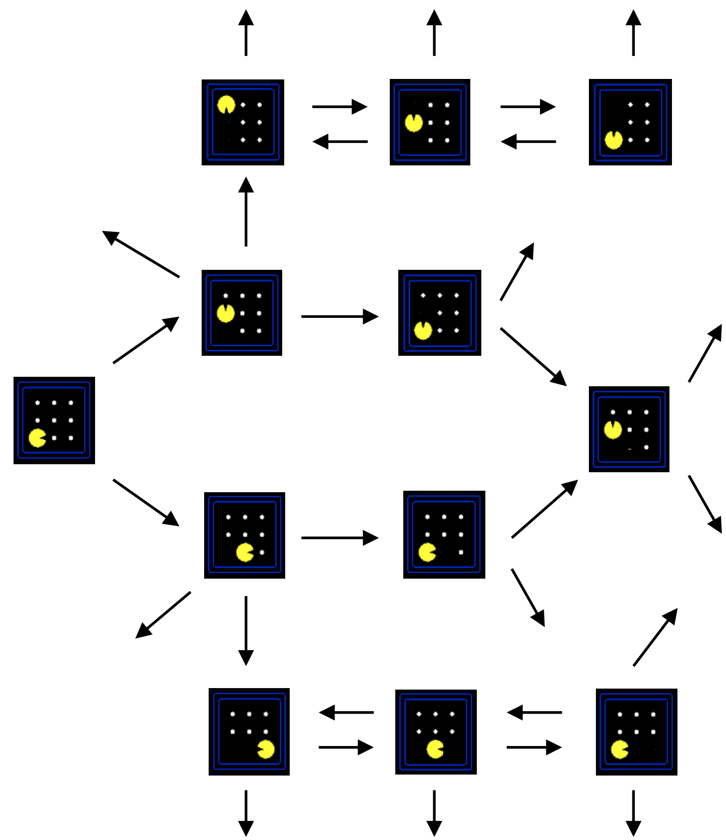
В.Бондарев

Граф пространства состояний

- Граф пространства состояний: математическое представление задачи поиска
  - Узлы фиксируют состояния задачи;
  - Дуги показывают приемников (результаты действий);
  - Целевая проверка – множество целевых состояний (возможно только одно).

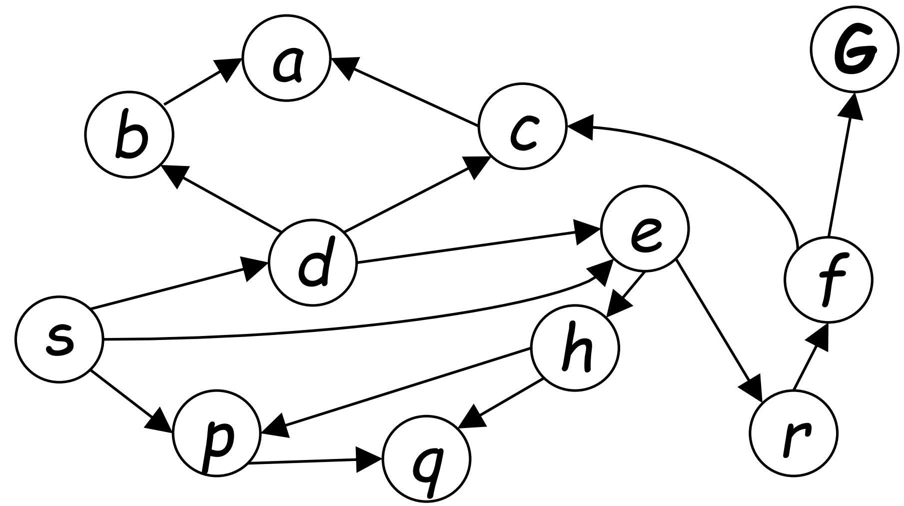
- На графе состояний каждое состояние может встретиться только один раз!

- Мы не всегда можем полностью построить граф в памяти (он слишком большой)



Граф пространства состояний

- Граф пространства состояний: математическое представление задачи поиска
  - Узлы фиксируют состояния задачи;
  - Дуги показывают приемников (результаты действий);
  - Целевая проверка – множество целевых состояний (возможно только одно).

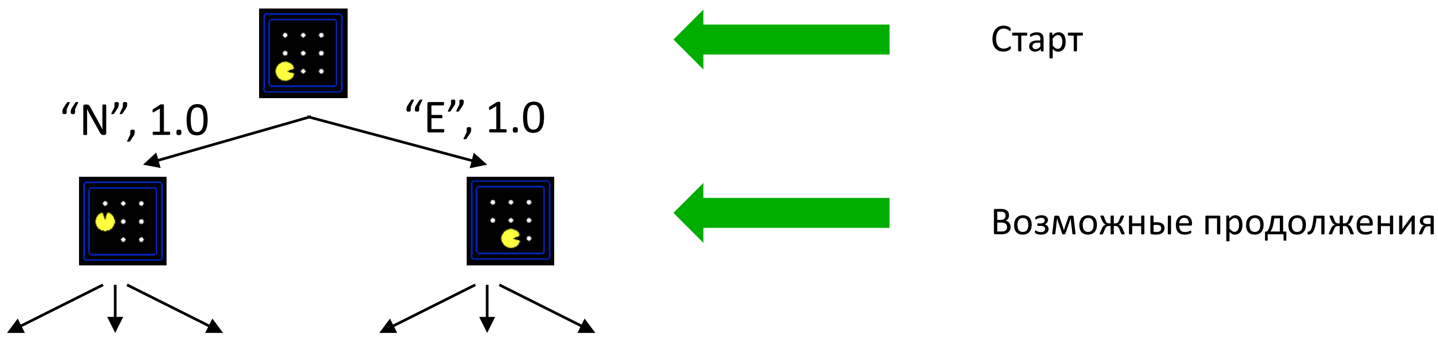


Небольшой граф состояний для простой задачи поиска

- На графе состояний каждое состояние может встретиться только один раз!

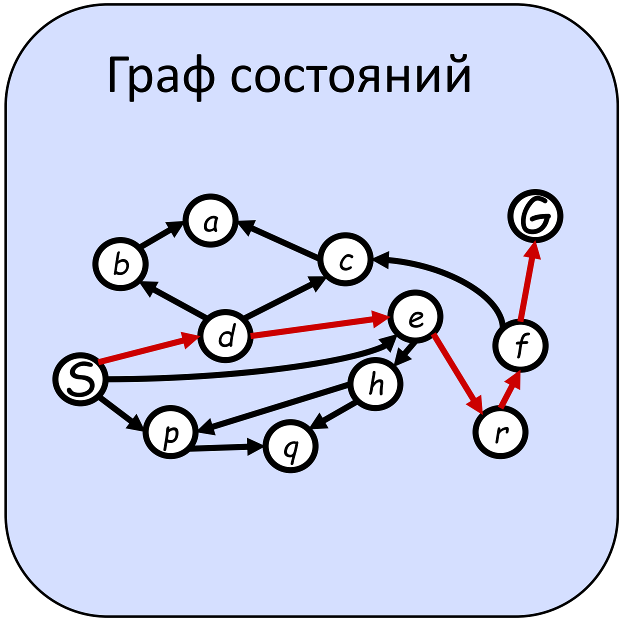
- Мы не всегда можем полностью построить граф в памяти (он слишком большой)

Деревья поиска



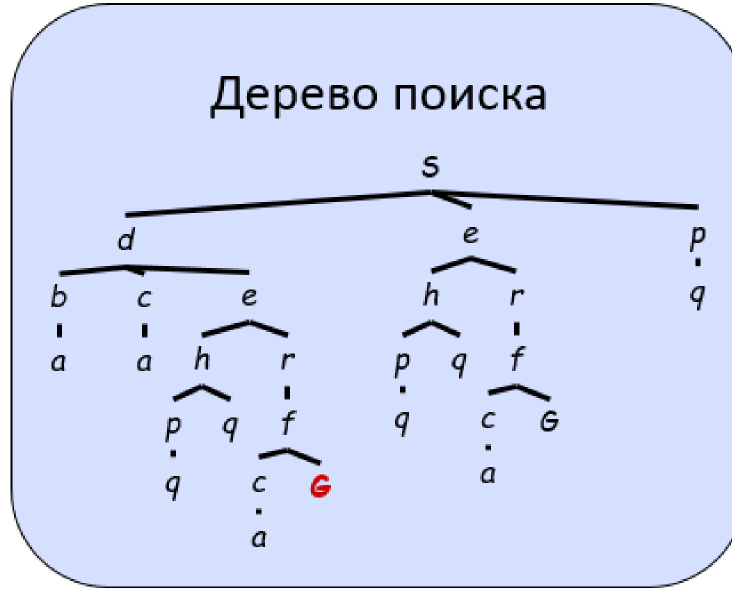
- Дерево поиска:
  - Дерево планов действий «что если» и их результатов;
  - Стартовое состояние является корнем дерева;
  - «Дети» соответствуют узлам приемникам ;
  - Узлы отображают состояния и соответствуют ПЛАНАМ, достижения этих состояний;
  - Для большинства задач мы не можем в действительности построить полное дерево

Граф состояний и дерево поиска



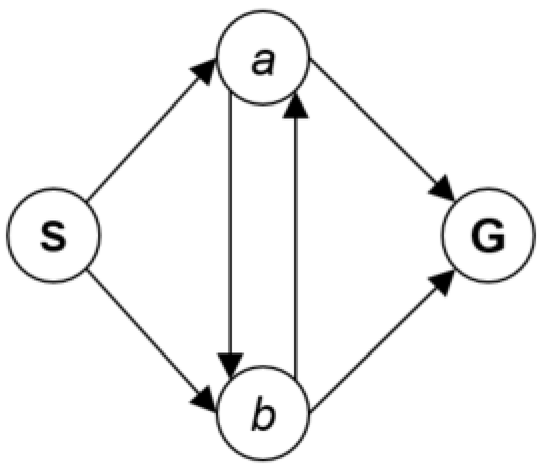
Каждый УЗЕЛ дерева поиска представляет собой целый ПУТЬ на графе пространства состояний.

Мы строим и то, и другое при необходимости, и строим в минимальном объеме

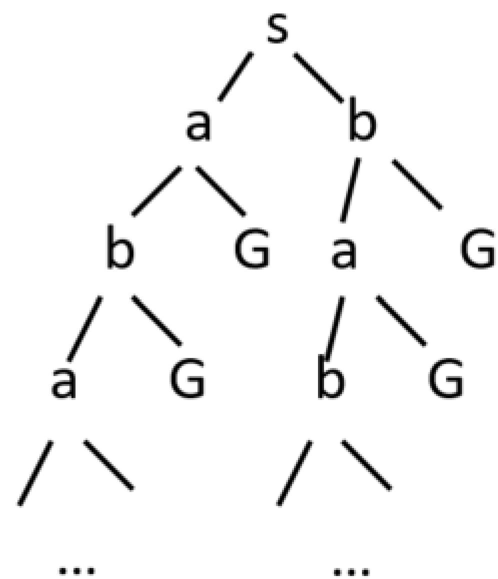


Quiz: **Граф состояний и дерево поиска**

Граф с 4-мя состояниями:



Каков размер его дерева поиска (при старте в S)?



∞

Важно: Множество повторяющихся структур на дереве поиска!

Поиск в ширину

Рассматриваемые ниже стратегии поиска используют два списка:

1. **Список открытых вершин** (**OPEN**) содержит идентификаторы (имена) вершин, подлежащих раскрытию;

2. **Список закрытых вершин** (**CLOSED**) содержит имена уже раскрытых вершин. Список **CLOSED** позволяет запоминать уже рассмотренные вершины с целью исключения их повторного раскрытия.

Согласно этой стратегии, при поиске по дереву вершины с глубиной ***k***, раскрываются после того как будут раскрыты все вершины глубиной ***k-1***. В этом случае фронт поиска растет в ширину. Для построения обратного пути (из целевой вершины в начальную вершину) все дочерние вершины снабжаются ссылками на соответствующие родительские вершины.

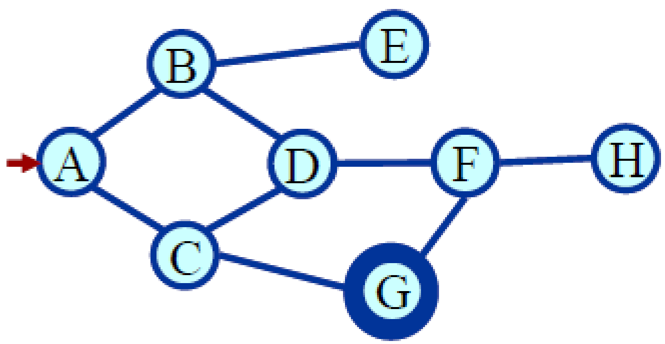
Поиск в ширину

```
Procedure Breadth_First_Search;   {BFS}
Begin
        Поместить начальную вершину в список OPEN;
        CLOSED:=‘пустой список’;
        While OPEN<>‘пустой список’ Do
        Begin
                n:=first(OPEN);
                If n=‘целевая вершина’ Then Выход(УСПЕХ);
                Переместить n из списка OPEN в CLOSED;
                Раскрыть вершину n и поместить все ее
                дочерние вершины [, которых нет в списках
                CLOSED и OPEN,] в конец списка OPEN,
                связав с каждой дочерней вершиной
                указатель на n;
        End;                                         В ЛР!
        Выход(НЕУДАЧА);                              OPEN = util.Queue()
End.
```

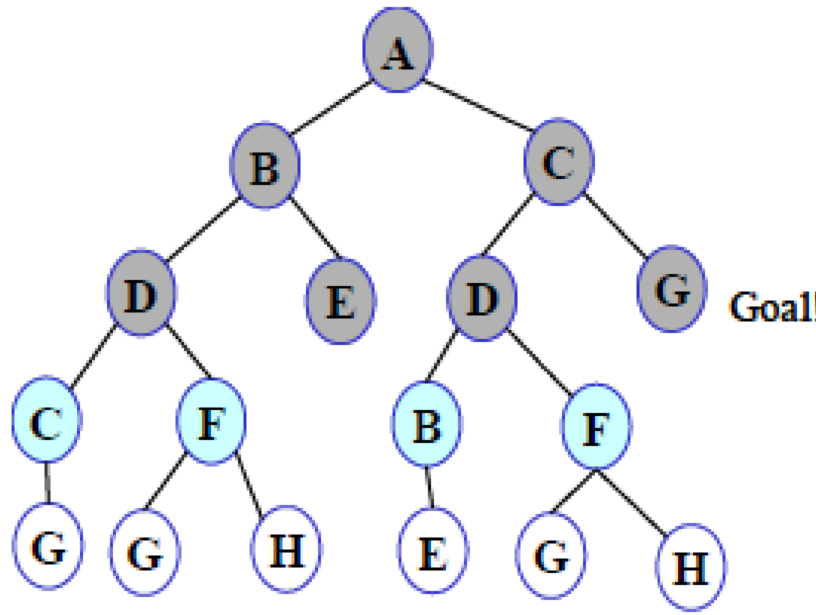
В.Бондарев

Поиск в ширину

Граф поиска



Дерево поиска



Если повторяющиеся узлы не исключаются:

OPEN (очередь FIFO)
1. (a.a)
2. (b.a c.a)
3. (c.a d.b e.b)
4. (d.b e.b d.c g.c)
5. (e.b d.c g.c c.d f.d)
6. (d.c g.c c.d f.d)
7. (g.c c.d f.d b.d f.d)

CLOSED
1. –
2. (a.a)
3. (b.a a.a)
4. (c.a b.a a.a)
5. (d.b c.a b.a a.a)
6. (e.b d.b c.a b.a a.a)
7. (d.c e.b d.b c.a b.a a.a)
8. (g.c d.c e.b d.b c.a b.a a.a)

Зеленым цветом отмечены повторяющиеся узлы

Решение: (G.C C.A A.A)

В.Бондарев

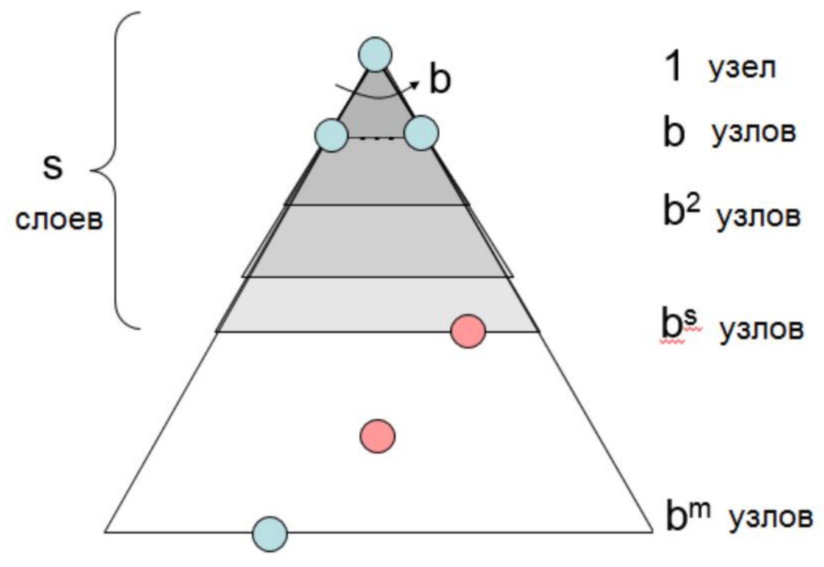
Поиск в ширину

Возможная структура данных узла:

1. Описание состояния;
2. Ссылка на родительский узел;
3. Счетчик глубины (стоимости) узла;
4. Оператор, сгенерировавший узел;

Свойства BFS

1. Временная и пространственная сложности.



1. Если повторяющиеся вершины не исключаются из рассмотрения (т.е. строится **дерево поиска**!) и каждая вершина имеет ***b*** дочерних вершин, то при остановке поиска на глубине, равной ***s***, максимальное число раскрытых вершин будет равно

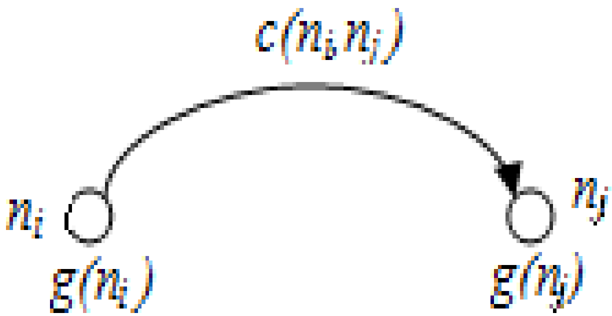
$$O(b^{s+1}) = 1+b+b^2+b^3+\ldots+b^s+(b^{s+1}-b)$$

Т.е. **оценки временной и пространственной** сложности являются ***экспоненциальными*** **-** $O(b^s)$. Поэтому поиск в ширину может использоваться только для задач с небольшой размерностью пространства состояний.

2. Является полным.

3. **Является оптимальным**, если все операторы имеют равную стоимость (обеспечивает нахождение решения, которое находится на минимальной глубине ***s***, т.е. *самого поверхностного решения*).

Поиск по критерию стоимости (алгоритм равных цен)



Оператору, преобразующему состояние n_i в состояние n_j, ставится в соответствие некоторая положительная функция стоимости $c(n_i, n_j)$. Тогда стоимость пути к вершине n_j может быть определена по формуле

$$g(n_j) = g(n_i) + c(n_i, n_j),$$

где $g(n_i)$ – стоимость пути из начальной вершины в вершину n_i.

Каждый раз из списка **OPEN** выбирается вершина с наименьшей стоимостью. Это позволяет найти путь минимальной стоимости из начальной вершины в целевую вершину. *Следует отметить, что в процессе поиска стоимость пути к вершине может меняться, если обнаруживается более дешевый путь.*

Алгоритм равных цен

Procedure Uniform_Cost_Search;
Begin
Поместить начальную вершину в список OPEN;
CLOSED:=‘пустой список’;
While OPEN<>‘пустой список’ Do Begin
n:=first(OPEN);
If n=‘целевая вершина’ Then Выход(УСПЕХ);
Переместить вершину n из списка OPEN в CLOSED;
Раскрыть вершину n, для каждой дочерней вершины *i* вычислить стоимость $\hat{g}(n, n_i)$ **;**
Поместить дочерние вершины, которых нет в списках CLOSED и OPEN, в список OPEN, связав с каждой вершиной указатель на вершину n и положить $\hat{g}(n_i) = \hat{g}(n, n_i)$ **;**

Алгоритм равных цен

Для каждой из дочерних вершин, которые уже содержатся в списке OPEN, сравнить текущую стоимость $\hat{g}(n, n_i)$ с ранее вычисленным значением стоимости $\hat{g}(n_i)$, хранящемся в списке OPEN, если $\hat{g}(n, n_i)$ < $\hat{g}(n_i)$, то установить $\hat{g}(n_i)$ = $\hat{g}(n, n_i)$. Снабдить указанные дочерние вершины указателями на вершину n;

Упорядочить список OPEN по возрастанию стоимости;

End;

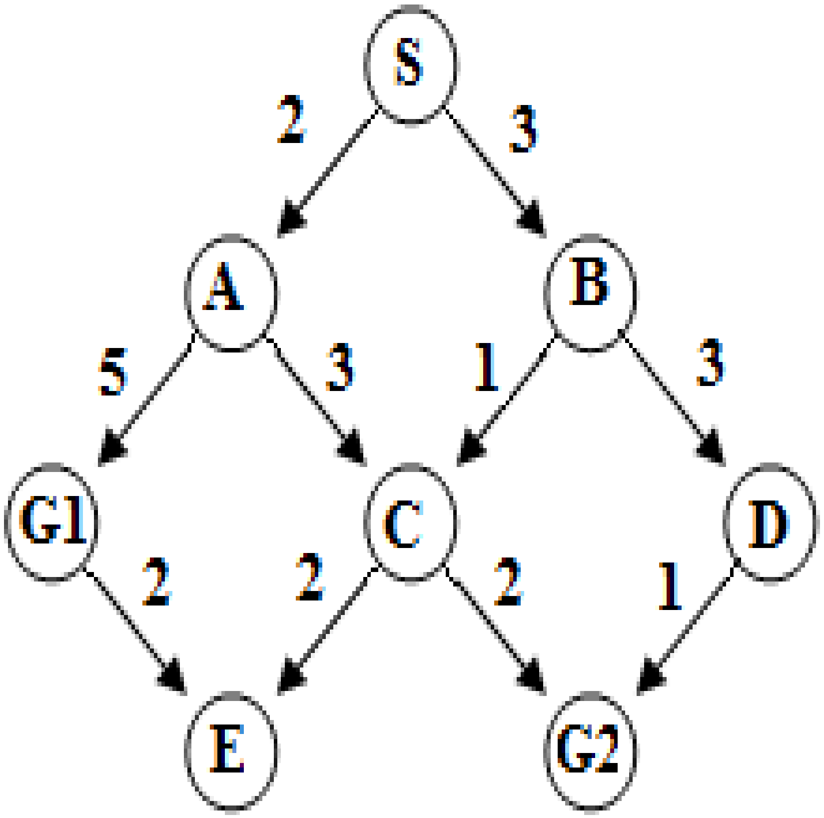
Выход(НЕУДАЧА);

End.

В ЛР!

OPEN = util.PriorityQueue()

Пример выполнения алгоритма равных цен



Список OPEN:

S(0)

A(2) B(3)

B(3) C(5) G1(7)

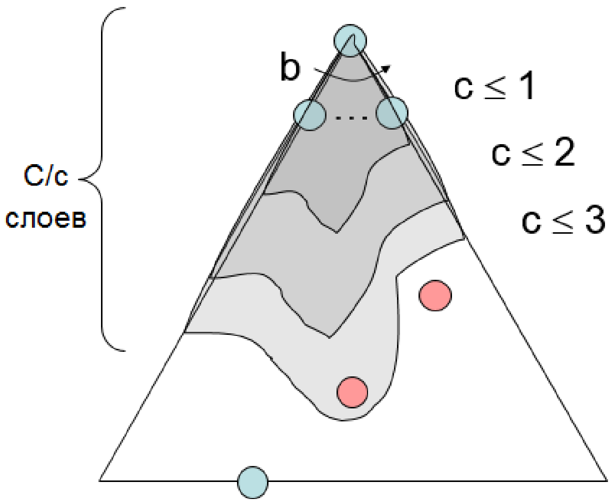
C(4) D(6) G1(7)

G2(6) D(6) E(6) G1(7)

Свойства алгоритма равных цен (UCS)

Теорема: *В тот момент, когда раскрывается вершина n оптимальный путь к этой вершине уже найден.*

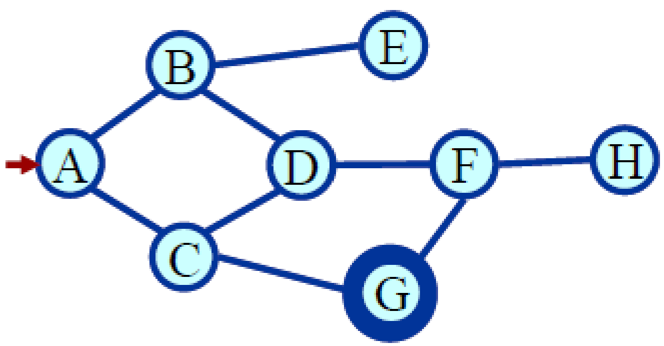
1. Рассмотренная стратегия гарантирует **полноту** поиска, если стоимость каждого участка пути положительная величина.

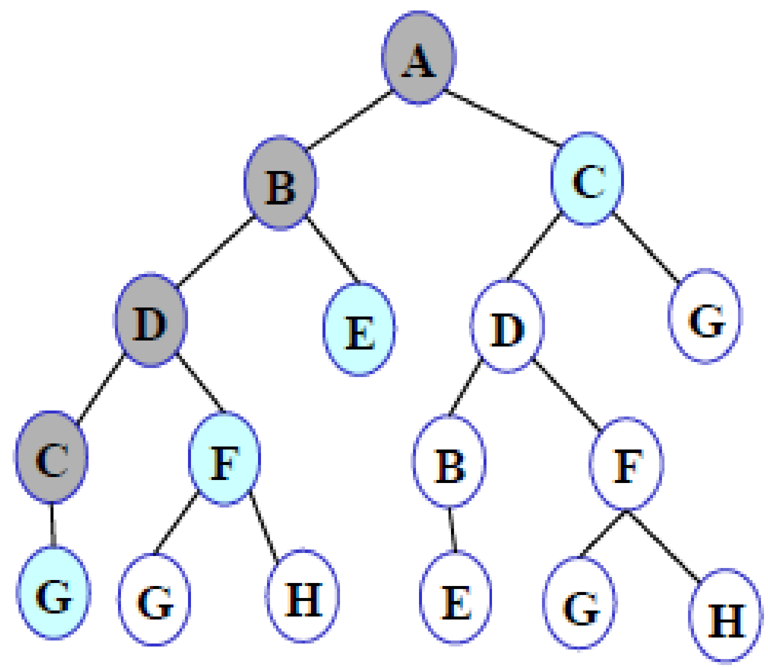


2. Так как поиск в этом случае направляется стоимостью путей, то оценки **временной и пространственной сложности** являются **экспоненциальными** и в наихудшем случае будут равны $O(b^{1+C/c})$, где C — стоимость оптимального решения, c — минимальная стоимость действия. Эта оценка может быть больше $O(b^s)$. Это связано с тем, что процедура поиска по критерию стоимости часто обследует поддеревья поиска, состоящие из ***мелких этапов небольшой стоимости***, прежде чем перейти к исследованию путей, в которые входят крупные, но возможно более полезные этапы .

Поиск в глубину (Depth First Search – DFS)

При поиске в глубину всегда раскрывается самая глубокая вершина в текущем фронте поиска. Процедура поиска в глубину отличается от рассмотренной процедуры поиска в ширину тем, что дочерние вершины, получаемые при раскрытии вершины **n**, помещаются в **начало** списка **OPEN** т.е. принцип формирования списка **OPEN** соответствует стеку (LIFO)





Решение: A-B-D-C-G

Поиск в глубину (Depth First Search – DFS)

Рассмотрим программирование DFS в среде Pacman.

Уточним возможные значения некоторых переменных в среде Pacman.

print("Start:", problem.getStartState())
print(«Является Start целевой?", problem.isGoalState(problem.getStartState()))
print(«Приемники Start:", problem.getSuccessors(problem.getStartState())

Start: (5, 5)
Является Start целевой ? False
Приемники Start: [((5, 4), 'South', 1), ((4, 5), 'West', 1)]

локация действие стоимость

В.Бондарев

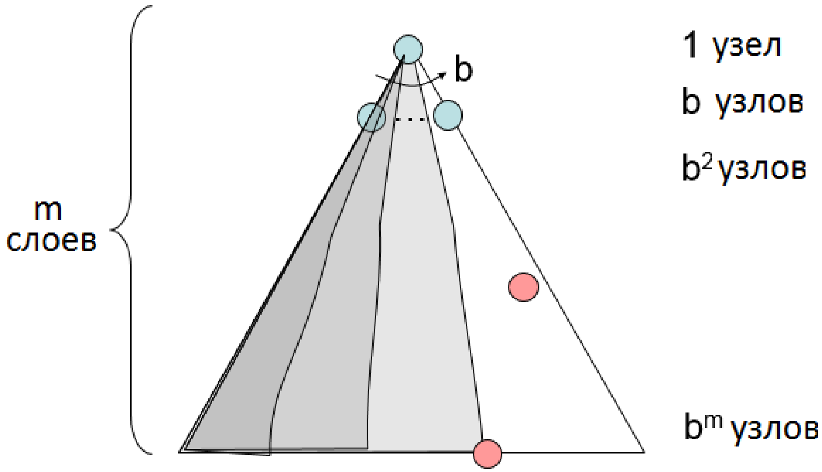
Поиск в глубину (Depth First Search – DFS)

```
# open - список-стек отрытых вершин
open=util.Stack()
# извлекаем из problem стартовую вершину ( getStartState() )
# и создаем стартовое состояние в виде списка - [старт-узел, стоимость, путь]
start=[problem.getStartState(), 0, [ ] ]
# вносим start  в начало списка открытых вершин
open.push(start)
# closed - список закрытых вершин
closed=[]
#цикл  - пока список отрытых вершин не пустой
while not open.isEmpty():
    # извлекаем первую вершину из списка открытых вершин
    [node, cost, path] =open.pop()
    # проверяем является ли она целевой
    if problem.isGoalState(node):
        # если да, то возвращаем путь до неё - список действий
        return path
    #если нет, то добавляем эту вершину в список closed
    closed.append(node)
    # раскрываем текущую вершину и получаем список доч.вершин
    successors=problem.getSuccessors(node)
```

Поиск в глубину (Depth First Search – DFS)

```
# для всех дочерних вершин
for child_node, child_act, child_cost in successors:
    # если дочерняя вершина не входит в список closed
    if (not child_node in closed):
        # то вычисляем стоимость её раскрытия
        new_cost = cost + child_cost
        # формируем путь к этой доч. вершине
        new_path = path + [child_act]
        # формируем новое состояние
        new_state = [child_node, new_cost, new_path]
        # вносим его в начало списка открытых вершин
        open.push(new_state)
```

Свойства алгоритма поиска в глубину



1. В наихудшем случае **временная сложность** (число раскрытых вершин) равна $O(b^m)$, где m — максимальная глубина дерева поиска (m может быть гораздо больше по сравнению с s — глубиной самого поверхностного решения).

2. Поиск в глубину требует хранения только единственного пути от корня до листового узла. Для дерева поиска с коэффициентом ветвления b и максимальной глубиной m поиск в глубину требует хранения $bm+1$ узлов, т.е оценка **пространственной сложности** равна $O(bm)$, что намного меньше по сравнению в рассмотренными выше стратегиями.

3. При **поиске в глубину с возвратами** требуется еще меньше памяти. В этом случае каждый раз формируется только одна из дочерних вершин и запоминается информация о том, какая вершина должна быть сформирована следующей. Таким образом, требуется только $O(m)$ ячеек памяти.

4. Однако поиск в глубину **не является полным** (в случае неограниченной глубины) и **не является оптимальным** (не обеспечивает гарантированное нахождение наиболее поверхностного целевого узла).

Поиск с ограничением глубины

Проблему деревьев поиска неограниченной глубины можно решить, предусматривая применение во время поиска заранее определенного ***предела глубины L.*** Это означает, что вершины на глубине ***L*** рассматриваются таким образом, как если бы они не имели дочерних вершин. Такая стратегия поиска называется **поиском с ограничением глубины**.

Однако при этом вводится дополнительный источник **неполноты**, если будет выбрано ***L<s***, т.е. самая поверхностная цель находится за пределами глубины.

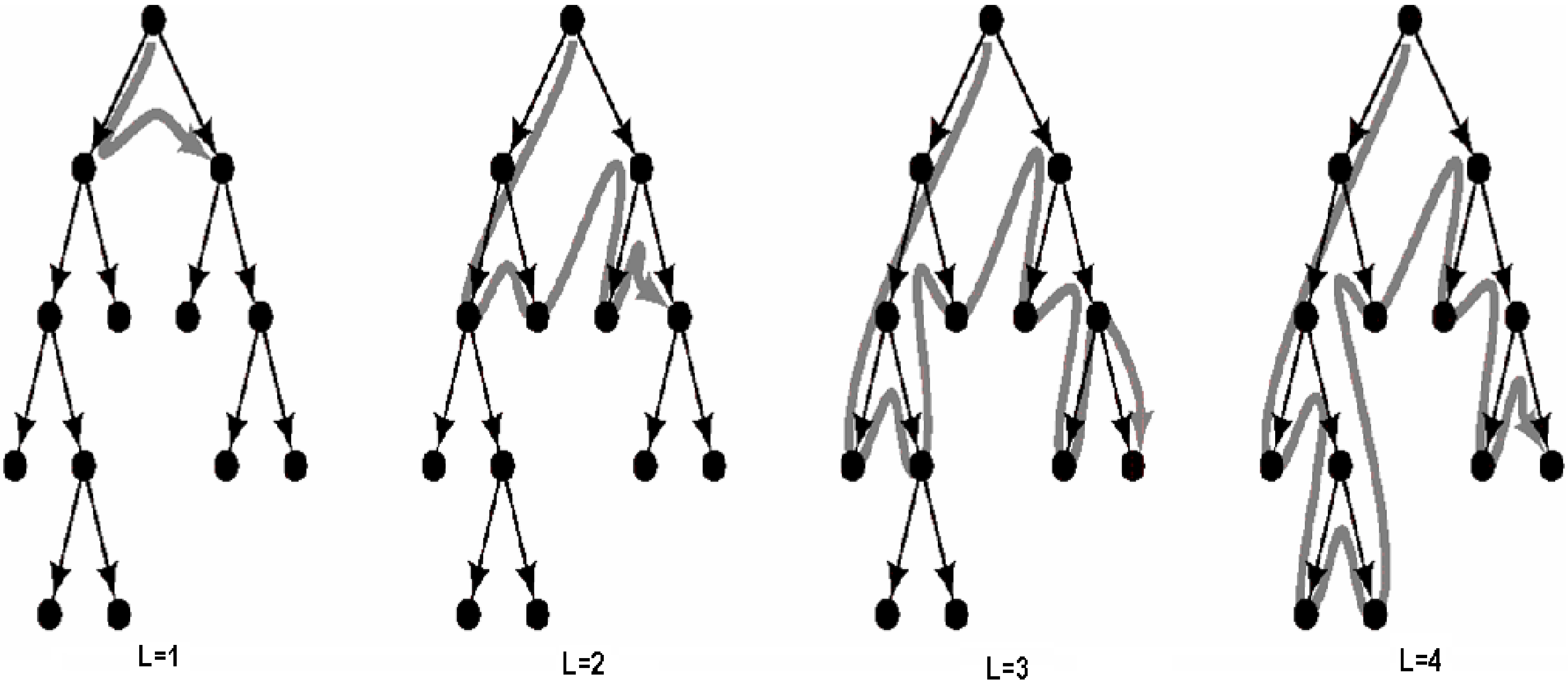
А при выборе ***L>s*** поиск с ограничением глубины будет **неоптимальным** (не гарантируется получение самого поверхностного решения).

Поиск с итеративным углублением (Depth-first iterative deeping – DFID)

Эта стратегия поиска позволяет найти наилучший предел глубины. Для этого применяется процедура поиска с ограничением по глубине. При этом предел глубины постепенно увеличивается (в начале он равен 0, затем 1, затем 2 и т.д.) до тех пор пока не будет найдена цель. Это происходит, когда предел глубины достигает значения ***s*** — глубины самого поверхностного решения.

Procedure DFID;
While *решение не найдено* Do begin
    ***Применить DFS с ограничением глубины уровнем* L;**
    L=L+1;
End;
End.

Поиск с итеративным углублением



Здесь вершины раскрываются повторно. Однако такие повторные операции не являются слишком дорогостоящими. Отношение числа вершин,   раскрываемых алгоритмом **DFID** по сравнению с **DFS,** равно **B/(B-1)**. Например, при **B=10** количество раскрываемых вершин будет больше только на **10%**

В.Бондарев

Св-ва поиска с итеративным углублением

1. **Полный.**
2. **Оптимальный**, если все операторы имеют одинаковую стоимость (гарантирует нахождение кратчайшего пути)
3. **Оценка временной сложности** несколько хуже, чем у **BFS** (так как вершины в верхней части дерева поиска раскрываются по нескольку раз). Если фактор ветвления равен $\boldsymbol{b}$ и решение находится на глубине $\boldsymbol{s}$, то вершины на этой глубине раскрываются один раз, на глубине $\boldsymbol{s-1}$ – два раза, и т.д., следовательно $\boldsymbol{b^s+2b^{(s-1)}+\ldots+sb \approx O(b^s)}$
4. **Пространственная сложность** – линейная, т.е. $\boldsymbol{O(bs)}$, подобно **DFS**

В поиске с итеративным углублением (по дереву поиска) сочетаются преимущества поиска в ширину (является **полным**) и поиска в глубину (малое значение **пространственной сложности**, равное $O(bs)$.

Сравнение методов слепого поиска

Критерий	BFS	DFS	DFID	Двунаправленный
Время	b^s	b^m	b^s	$b^{m/2}$
Память	b^s	bm	bs	$b^{m/2}$
Оптимальность	Да	Нет	Да	Да
Полнота	Да	Нет	Да	Да

Двунаправленный поиск возможен на неориентированных графах и предполагает движение из начального состояния в целевое и из целевого в исходное. Поиск прекращается, когда фронты поиска пересекутся (критерий: текущая вершина принадлежит другому дереву поиска).

Особенности слепого поиска на графах

Так как в рассмотренной выше обобщенной процедуре поиска в ширину для исключения повторяющихся состояний используется список **CLOSED**, в котором запоминаются уже раскрытые вершины, то **временная сложность** рассмотренных алгоритмов при **поиске на графе состояний** может быть меньше, чем при **поиске на дереве состояний**. Но из-за того, что запоминаются все рассмотренные состояния, то поиск в глубину и поиск с итеративным углублением уже не будут характеризоваться линейными оценками **пространственной сложности**. И рассмотренные методы поиска в этом случае могут оказаться неосуществимыми из-за недостаточного объема памяти.

В этом заключается фундаментальный компромисс между пространством и временем.