Posts from m.zhang

keep growing with every tiny trial

- RSS

Search

Navigate…

- Blog
- Archives
- Categories
- About

Dynamic Programming with Bitmask

Apr 20th, 2014 | Comments

DP is a technique to avoid repetitive computing by using a memo to track the result of each subproblem. In some scenarios, the state of a subproblem can be represented as a bitmask, and then the memo becomes an array. For the properties of bitmask, it fits the problem whose subproblems are defined on the subsets of variables. This post shows some scenarios using DP on bitmask I have ever met.

1. Baseline Models

1.1 Set Cover Problem

A classic NP-Complete problem in combinatorics and OR. According to SCP WIKI, the problem is defined as: Given a set of elements $\{1, 2, \ldots, m\}$, the "universe", and a set S of n set whose union equals to the universe, the problem is to identify the smallest subset of S whose union equals to the universe.

Here's a formulation,

$$
\begin{aligned}
& min \sum_{s \in S} cost_s x_s \\
& s.t. \\
& \quad \sum_{s:e \in s} x_s \geq 1 \qquad \text{for all } e \in U \\
& \quad x_s \in \{0, 1\} \qquad \text{for all } s \in S
\end{aligned}
$$

Enumerating and checking all subsets of S needs 2^n operations, while DP method costs $n2^m$. Thus, DP is efficient when the universe is relatively small.

1.2 TSP

Problem Definition: given a weighted graph with n nodes, find the shortest path that visits every node exactly once.

TSP is also a classic NP-Complete problem. Brute force method has the complexity of $O(n!)$, while DP method can reduce it to $O(n^2 2^n)$. The optimal substructure of the problem is,

$$D_{S,v} = \min_{u \in S-\{v\}} (D_{S-\{v\},u} + cost(u, v))$$

where the $D_{S,v}$ denotes the length of the optimal path that visits every node in S exactly once and ends at v.

Thus, there're $n2^n$ subproblems. The answer to the problem is $\min_{v \in V}(D_{V,v})$, where V is the complete set of nodes.

Reference: slides from Standford CS97SI.

2. Varieties and Examples

2.1 Set Covering

Codeforces 417D. A basic set covering problem, where the objective function brings a little difficulty.

417D.cc

```
/*
 * Codeforces 417D Cunning Gena.
 * Tags: dp bitmask sortings greedy
 * Date: April 21 2014
 * Author: mzhang
 */

#include <iostream>
#include <algorithm>

using namespace std;

#define N 100   // max friends number
#define M 20    // max problems number
#define INF -1

typedef long long ll;

struct Friend {
    int id;     // friend id.
    ll cost;    // the cost of solving problems.
    ll monitor; // the necessary num of monitors.
    int solves; // the bitmask of solvable problems.
};

bool cmp(Friend a, Friend b) {
    return a.monitor < b.monitor;
}

Friend friends[N];
ll dp[1<<M];

int main() {
    int n, m;   // number of friends and problems.
    ll b;       // cost of monitor.
    cin >> n >> m >> b;
```

```
37
38      for(int i = 0; i < (1<<m); i++) {
39          dp[i] = INF; // initially marked as impossible.
40      }
41      dp[0] = 0;
42
43      /* input processing. */
44      for(int i = 0; i < n; i++) {
45          ll cost, monitor, num;
46          int solves = 0;
47          cin >> cost >> monitor >> num;
48          for(int j = 0; j < num; j++) {
49              int qid;
50              cin >> qid;
51              solves |= (1 << (qid-1));
52          }
53          friends[i] = (Friend){i+1, cost, monitor, solves};
54      }
55
56      // start with the friend with minimum monitor number.
57      sort(friends, friends+n, cmp);
58
59      ll min_cost = INF;
60      for(int fid = 0; fid < n; fid++) {
61          Friend f = friends[fid];
62          for(int i = 0; i < (1<<m); i++) {
63              if(dp[i] == INF) {
64                  continue;
65              }
66              if(dp[i | f.solves] == INF) {
67                  dp[i | f.solves] = dp[i] + f.cost;
68              }
69              else {
70                  dp[i | f.solves] = min(dp[i | f.solves], dp[i] + f.cost);
71              }
72          }
73          if(dp[(1<<m) - 1] != INF) {
74              if(min_cost == INF) {
75                  min_cost = (dp[(1<<m) - 1] + f.monitor * b);
76              }
77              else {
78                  min_cost = min(min_cost, dp[(1<<m) - 1] + f.monitor * b);
79              }
80          }
81      }
82
83      cout << min_cost << endl;
84  }
```

2.2 TSP

POJ 3311. A basic TSP problem, however it requires the path start and end with the central point. In addition, floyd algorithm is needed to preprocess the distance matrix.

poj_3311.cc

```
1  /*
2   * POJ 3311 Hie with the Pie.
3   * Date: April 22 2014
4   * Author: mzhang
5   * Tag: dp bitmask TSP
6   */
```

```
7
8  #include <iostream>
9
10 using namespace std;
11
12 #define N 11        // max nodes number
13 #define INF -1      //
14 #define S -1        // id of the starting and ending node
15
16 int dp[1<<N][N];
17 int d[N][N];        // distance matrix
18
19 void input(int d[N][N], int n) {
20     for(int i = 0; i < n; i++) {
21         for(int j = 0; j < n; j++) {
22             cin >> d[i][j];
23         }
24     }
25 }
26
27 bool is_element(int bitmask, int index) {
28     return bitmask == (bitmask | (1<<index));
29 }
30
31 int dist(int i, int j) {
32     return d[i+1][j+1];
33 }
34
35 void floyd(int d[N][N], int n) {
36     for(int i = 0; i < n; i++) {
37         for(int j = 0; j < n; j++) {
38             for(int k = 0; k < n; k++) {
39                 d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
40             }
41         }
42     }
43 }
44
45 int main() {
46     int n;
47     while(cin >> n) {
48         if(n == 0) {
49             break;
50         }
51
52         /* get distance matrix. */
53         input(d, n+1);
54
55         /* distance matrix preprocess. */
56         floyd(d, n+1);
57
58         /* dp init.
59          * All the path should start with the S. */
60         for(int i = 0; i < n; i++) {
61             for(int k = 0; k < (1<<n); k++) {
62                 dp[k][i] = INF;
63             }
64             dp[1<<i][i] = dist(S, i);
65         }
66
67         /* dp stage. */
68         for(int i = 1; i < (1<<n); i++) {
69             for(int j = 0; j < n; j++) {
70                 if(dp[i][j] != INF) {
71                     continue;
```

```
72                }
73                if(is_element(i, j)) {
74                    int min_dist = INF;
75                    for(int k = 0; k < n; k++) {
76                        if(is_element(i, k) && k != j) {
77                            if(min_dist == INF ||
78                                    min_dist > dp[i-(1<<j)][k] + dist(k, j)) {
79                                min_dist = dp[i-(1<<j)][k] + dist(k, j);
80                            }
81                        }
82                    }
83                    dp[i][j] = min_dist;
84                }
85            }
86        }
87
88        /* Choosing final result.
89         * The complete path need to end with S */
90        int min_dist = INF;
91        for(int i = 0; i < n; i++) {
92            if(min_dist == INF || min_dist > dp[(1<<n)-1][i] + dist(i, S)) {
93                min_dist = dp[(1<<n)-1][i] + dist(i, S);
94            }
95        }
96        cout << min_dist << endl;
97    } // testcase loop
98 }
```

Codeforces 8C. The model and algorithm in TSP is applicable here if time and space complexity issue is not considered. This problem requires the “salesman” able to vist only 2 customers at most in a single trip from the start point. Thus, the trace of the optimal solution does not need to be “strongly sorted”. Based on this property, the model and algorithm can be simplify.

8C.cc

```
1  /*
2   * Codeforces 8C Looking for Order.
3   * Date: April 26 2014
4   * Author: mzhang
5   * Tag: dp bitmask
6   */
7
8  #include <iostream>
9  #include <cmath>
10 #include <cstdio>
11
12 #define N 24
13 #define INF -1
14 #define S -1
15
16 using namespace std;
17
18 struct Point {
19     int x, y;
20 };
21
22 Point points[N+1];
23 int d[N+1][N+1];
24 int dp[1<<N];
25 int pre[1<<N];
26 int n;
27
```

```
28 int dist(int i, int j) {
29     return d[i+1][j+1];
30 }
31
32 void get_dist_mat(int d[N+1][N+1], Point points[N+1], int n) {
33     for(int i = 0; i < n; i++) {
34         for(int j = 0; j < n; j++) {
35             d[i][j] = pow(points[j].x - points[i].x, 2) +
36                 pow(points[j].y - points[i].y, 2);
37         }
38     }
39 }
40
41 int count(int bitmask) {
42     int cnt = 0;
43     for(int i = 0; i < n; i++) {
44         if(bitmask & (1<<i)) {
45             cnt++;
46         }
47     }
48     return cnt;
49 }
50
51 bool is_element(int bitmask, int i) {
52     return (bitmask == (bitmask | (1<<i)));
53 }
54
55 void get_pre_nodes(int cur_state, int p_state, int pnodes[2], int& pcnt) {
56     int diff = cur_state ^ p_state;
57     pcnt = 0;
58     for(int i = 0; i < n; i++) {
59         if(diff & (1<<i)) {
60             pnodes[pcnt] = i;
61             pcnt++;
62         }
63     }
64     return;
65 }
66
67 /* compare and update first arg.
68  */
69 int get_min(int& x, int y) {
70     if(x == INF) {
71         x = y;
72         return true;
73     }
74     if(y == INF) {
75         return false;
76     }
77     if(x < y) {
78         return false;
79     }
80     x = y;
81     return true;
82 }
83
84 int main() {
85     cin >> points[0].x >> points[0].y;
86     cin >> n;
87     for(int i = 1; i <= n; i++) {
88         cin >> points[i].x >> points[i].y;
89     }
90     get_dist_mat(d, points, n+1);
91
92     for(int i = 0; i < (1<<n); i++) {
```

```
93          dp[i]= INF;
94      }
95      dp[0]= 0;
96
97      for(int i = 1; i < (1<<n); i++) {
98          int min_dist = INF;
99          // pick one.
100         for(int j = 0; j < n; j++) {
101             if(!is_element(i, j)) {
102                 continue;
103             }
104             bool modified = get_min(min_dist, dp[i-(1<<j)] + dist(S, j)*2);
105             if(modified) {
106                 pre[i] = i - (1<<j);
107             }
108         }
109         // pick two, if possible.
110         if(count(i) <= 1) {
111             dp[i] = min_dist;
112             continue;
113         }
114         for(int j = 0; j < n-1; j++) {
115             if(is_element(i, j)) {
116                 for(int k = j+1; k < n; k++) {
117                     if((!is_element(i, j)) || (!is_element(i, k))) {
118                         continue;
119                     }
120                     bool modified = get_min(min_dist, dp[i-(1<<j)-(1<<k)] +
121                             dist(S, j) + dist(j, k) + dist(k, S));
122                     if(modified) {
123                         pre[i] = i - (1<<j) - (1<<k);
124                     }
125                 }
126                 break;
127             }
128         }
129         dp[i] = min_dist;
130     }
131     printf("%d\n", dp[(1<<n)-1]);
132
133     printf("0");
134     int cur = (1<<n)-1;
135     while(cur) {
136         int p = pre[cur];
137         int pnodes[2];
138         int pcnt = -1;
139         get_pre_nodes(cur, p, pnodes, pcnt);
140         if(pcnt == 1) {
141             printf(" %d 0", pnodes[0]+1);
142         }
143         else {
144             printf(" %d %d 0", pnodes[0]+1, pnodes[1]+1);
145         }
146         cur = p;
147     }
148     printf("\n");
149 }
```

2.3 Others

poj_3254.cc

```
1  /* POJ 3254 Corn Fields
2   * Date: April 27 2014
3   * Author: mzhang
4   * Tags: dp bitmask
5   */
6
7  #include <iostream>
8
9  using namespace std;
10
11 #define N 12
12 #define M 12
13 #define INVALID -1
14
15 #define MOD 1000000000
16
17 int n, m;
18 int map[N];
19 int dp[N][1<<M];
20
21 /* judge the validity of a row pattern,
22  * not considering the pre and post lines.
23  */
24 bool valid(int r, int bitmask) {
25     return (bitmask == (bitmask & map[r])) && !(bitmask & (bitmask << 1));
26 }
27
28 /* judge the validity of a row pattern in the presence of a previous line.
29  */
30 bool conflict(int pre, int cur) {
31     return pre & cur;
32 }
33
34 int main() {
35     scanf("%d%d", &n, &m);
36
37     /* input preprocessing */
38     for(int i = 0; i < n; i++) {
39         map[i] = 0;
40         for(int j = 0; j < m; j++) {
41             int valid;
42             scanf("%d", &valid);
43             if(valid) {
44                 map[i] |= (1<<j);
45             }
46         }
47     }
48
49     /* init first row. */
50     for(int i = 0; i < (1<<m); i++) {
51         if(valid(0, i)) {
52             dp[0][i] = 1;
53         }
54         else {
55             dp[0][i] = INVALID;
56         }
57     }
58
59     /* DP phase */
60     for(int i = 1; i < n; i++) {
61         for(int j = 0; j < (1<<m); j++) {
62             dp[i][j] = INVALID;
63             if(!valid(i, j)) {
64                 continue;
65             }
```

```
66              for(int pre = 0; pre < (1<<m); pre++) {
67                  if(dp[i-1][pre] == INVALID) {
68                      continue;
69                  }
70                  if(conflict(pre, j)) {
71                      continue;
72                  }
73                  if(dp[i][j] == INVALID) {
74                      dp[i][j] = 0;
75                  }
76                  dp[i][j] += dp[i-1][pre];
77                  if(dp[i][j] > MOD) {
78                      dp[i][j] -= MOD;
79                  }
80              }
81          }
82      }
83
84      /* count */
85      int cnt = 0;
86      for(int i = 0; i < (1<<m); i++) {
87          if(dp[n-1][i] != INVALID) {
88              cnt += dp[n-1][i];
89              if(cnt > MOD) {
90                  cnt -= MOD;
91              }
92          }
93      }
94      printf("%d\n", cnt);
95  }
```

Posted by mengyu zhang Apr 20th, 2014 algorithm bitmask, dp

« The Last Program written in Baidu Classic DP Problems »

Comments

Recent Posts

- Codeforces Report Goodbye 2014
- HackerRank Lambda Calculi 09
- Codeforces Monthly Report 2014/07
- Design Patterns in Real Work
- Linux Kernel Experiment: Add System Call

Tags

b-tree, baidu, bitmask, bochs, codeforces, consistency, dp, fp, gevent, grub, hackerrank, interview, kernel, libevent, linux, memcached, python, qemu, rpc, zmq, zookeeper

GitHub Repos

- lambda-calculi

  Hackerrank Lambda Calculi

- mpcs51300-compiler

  MPCS51300 Compiler 2015 Fall

- scala-segtree

  Immutable Segment Tree in Scala.

- elogging

  Extension for Logging.

- hackerrank-scala

  Hackerrank practices in Scala.

- plenario-analysis

  experiments on various backend of plenario

- courses

  Collection of course exercises and projects

- emulab-scripts

  scripts for remote operations

- algorithm-training

  Samples and solutions of algorithm problems.

- articlist

  Personal blogs hosted on Heroku

- land-of-fp

  Practices on various LISP and ML dialects.

- my-zhang.github.io

  Personal Page

- nand2tetris

  Project solutions of Nand2Tetris

- building-blocks

  some building blocks for future use

@my-zhang on GitHub