算法设计与分析笔记

stable matching

matching: 每个人有0或1个对象与之匹配

perfect matching: 每个人都有一个对象与之匹配

unstable pair: $(m, w), \exists(m', w'), m$ perfer w' to $w \wedge w'$ perfer m to m'

stable matching: 没有unstable pair的perfect matching

n对n matching stable matching一定存在，但是不一定是唯一的，2n个人的stable matching可能不存在

Gale-Shapley Algorithm

```
initiaize eache person to be free
while (exists man not paired):
    choose such a man m
    find w = topest women m has not proposed to yet
    if (w is free):
        (m, w) become paired
    else if (w prefers m to her current partner m'):
        (m, w) become paired
        m' become free
    else:
        w rejects m
```

time complexity: $O(n^2)$
G-S ends after at most n^2 iterations
G-S finds a man-optimistic and women-pessimistic stable matching

1. Interval scheduling: $nlog(n)$ greedy algorithm.
2. Weighted interval scheduling: $nlog(n)$ dynamic programming algorithm.
3. Bipartite matching: n^k max-flow based algorithm.
4. Independent set: NP-complete.
5. Competitive facility location: PSPACE-complete.

Algorithm Analysis

upper bound:
$f(n)$ is $O(g(n))$ if there exist $c > 0$ and $n_0 > 0$ such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0$

lower bound::
$f(n)$ is $\Omega(g(n))$ if there exist $c > 0$ and $n_0 > 0$ such that $0 \leq c \cdot g(n) \leq f(n)$ for all $n \geq n_0$

tight bound:
$f(n)$ is $\Theta(g(n))$ if there exist $c_1, c_2 > 0$ and $n_0 > 0$ such that $0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ for all $n \geq n_0$

properties:

1. Reflexivity: $f = O(f)$

2. Constants: if f is O(g) and c > 0, then $c \cdot f$ is $O(g(n))$
3. Products: if f_1 is $O(g_1)$ and f_2 is $O(g_2)$, then $f_1 \cdot f_2$ is $O(g_1 \cdot g_2)$
4. Sums: if f_1 is $O(g_1)$ and f_2 is $O(g_2)$, then $f_1 + f_2$ is $O(max(g_1, g_2))$
5. Transitivity: if f is $O(g)$ and g is $O(h)$), then f is $O(h)$

$f(n)$ is $\Theta(g(n))$ iff $f(n)$ is $O(g(n))$ and $\Omega(g(n))$
if $lim_{n\to\infty}\frac{f(n)}{g(n)} = c$, then $f(n)$ is $\Theta(g(n))$
if $lim_{n\to\infty}\frac{f(n)}{g(n)} = 0$, then $f(n)$ is $O(g(n))$
if $lim_{n\to\infty}\frac{f(n)}{g(n)} = \infty$, then $f(n)$ is $\Omega(g(n))$

asymptotic bounds of common functions:

1. $f(n) = a_0 + a_1 \cdot n + a_2 \cdot n^2 + ... + a_d \cdot n^d$, where $a_d > 0$, then $f(n)$ is $\Theta(n^d)$
2. $log_a(n)$ is $\Theta(log_b(n))$ for any $a, b > 1$
3. $log_a(n)$ is $O(n^k)$ for any $k > 0$ and $a > 1$
4. r^n is $\Omega(n^k)$ for any $k > 0$ and $r > 1$
5. $n!$ is $2^{\Theta(nlog(n))}$ Proof: $n! \sim \sqrt{2\pi n}(\frac{n}{e})^n$

multiple variables:
$f(m,n)$ is $O(g(m,n))$ if there exist $c > 0$ and $n_0 > 0$ and $m_0 > 0$ such that $0 \leq f(m,n) \leq c \cdot g(m,n)$ for all $n \geq n_0$ and $m \geq m_0$
e.g. $f(m,n) = 32mn^2 + 17mn + 32n^3$
$f(m,n)$ is $O(mn^2 + n^3)$ and $O(mn^3)$ but not $O(n^3)$ nor $O(mn^2)$
f(m,n) is $O(n^3)$ if $m \geq n$ is implied.

sorting is $O(nlog(n))$
merge two sorted lists is $O(n)$
target sum on sorted list is $O(n)$
three sum on sorted list is $O(n^2)$
closest pair of points on a plane is $O(nlog(n))$
find disjoint sets in n sets is $O(n^3)$
independent set of size k is $O(n^k)$
find maximum size of independent set is $O(n^2 2^n)$

图

图的表示

adjency matrix:
$A[i,j] = 1$ if $(i,j) \in E$ else 0
space: $O(n^2)$
check if $(i,j) \in E$: $O(1)$
enumerate all edges: $O(n^2)$

adjency list:
for each vertex v, store a list of vertices w such that $(v,w) \in E$
space: $O(n+m)$
check if $(i,j) \in E$: $O(deg(i))$
enumerate all edges: $O(n+m)$

path: a sequence of vertices $v_1, v_2, ..., v_k$ such that $(v_i, v_{i+1}) \in E$ for all $i = 1, 2, ..., k-1$

a path is **simple** if all vertices are distinct
a path is **cycle** if $v_1 = v_k$, and no other vertices are repeated
a cycle is **simple** if all vertices are distinct except $v_1 = v_k$
a graph is **connected** if there is a path between every pair of vertices
an undirected graph is a **tree** if it is connected and has no cycles
let G be a graph with n vertices, if G is connected, does not contain a cycle and has n-1 edges, then G is a tree.

BFS

```
push s into Q
while Q is not empty
    v = pop_front Q
    for each edge (v,w) in E
        if w is not visited
            push w into Q
            mark w as visited
```

time: $O(n+m)$

DFS

```
push s into S
while S is not empty
    v = pop S
    for each edge (v,w) in E
        if w is not visited
            push w into S
            mark w as visited
```

Testing Bipartiteness

bipartite graph cannot have odd cycle

```
push s into Q
color[s] = 0
while Q is not empty
    v = pop_front Q
    for each edge (v,w) in E
        if w is not visited
            push w into Q
            color[w] = 1 - color[v]
        else if color[w] == color[v]
            return false
```

Strongly Connected Graph

a directed graph is **strongly connected** if there is a path between every pair of vertices
check if a directed graph is strongly connected:

```
pick a vertex s
run DFS from s on G
if there is a vertex v not visited
    return false
run DFS from s on G^T
if there is a vertex v not visited
    return false
return true
```

Time: $O(n+m)$

DAG and Topological Sort

a directed graph is a **DAG** if it does not contain a cycle
topological sort: a linear ordering of vertices such that if $(u,v) \in E$, then u appears before v in the ordering
topological sort on DAG:

```
initialize in_degree[v] for all v
push all v such that in_degree[v] = 0 into Q
while Q is not empty
    v = pop_front Q
    output v
    for each edge (v,w) in E
        in_degree[w] -= 1
        if in_degree[w] == 0
            push w into Q
```

Greedy Algorithms

Greedy algorithms works if

1. greedy choice property: a globally optimal solution can be arrived at by making a locally optimal (greedy) choice
2. optimal substructure: an optimal solution to the problem contains within it optimal solutions to subproblems
   贪心算法有效的条件是：
3. 贪心选择性：每一步贪心选出来的一定是原问题的最优解的一部分
4. 最优子结构：每一步贪心选择之后，贪心解和剩余子问题的最优解构成了原问题的最优解

Interval Scheduling

interval scheduling problem: given a set of intervals, find a maximum-size subset of mutually compatible intervals
Algorithm: Earliest Finish Time First

Interval Partitioning

interval partitioning problem: given a set of intervals, find a minimum-size set of bins such that every interval is assigned to a bin and no two intervals assigned to the same bin overlap
Algorithm: Earliest Start Time First

scheduling to minimize lateness

scheduling to minimize lateness: given a set of jobs, each with a deadline and a processing time, find a schedule that minimizes the maximum lateness
Algorithm: Earliest Deadline First

Time: all above is $O(nlog(n))$

Optimal Caching

optimal caching: given a sequence of requests for data, find a cache of size k that minimizes the number of cache misses
Offline algorithms:

1. LIFO/FIFO
2. LRU (least recently used)

3. LFU (least frequently used)
4. **Optimal: FF (furthest in the future)**

Shortest Path

single-source shortest path problem: given a weighted graph G and a source vertex s, find a shortest path from s to every other vertex in G

dijsktra's algorithm:

```
initialize dist[v] = infinity for all v
dist[s] = 0
unvisited = V
while unvisited is not empty
    v = vertex in unvisited with min dist[v]
    remove v from unvisited
    for each edge (v,w) in E
        if dist[w] > dist[v] + weight(v,w)
            dist[w] = dist[v] + weight(v,w)
```

time: $O(n^2)$

Minimum Spanning Tree

minimum spanning tree: given a weighted graph G, find a spanning tree of G with minimum total weight
Algorithm: Prim's Algorithm

```
S = {s} for some s in V
T = {}
Repeat n-1 times
    find edge (v,w) with min weight such that v in S and w not in S
    add (v,w) to T
    add w to S
```

time: $O(mlog(n))$
Algorithm: Kruskal's Algorithm

```
T = {}
sort edges by weight
make set for each vertex
for each edge (v,w) in E
    if v and w are in different sets
        add (v,w) to T
        merge sets containing v and w
```

divide and conquer

$$T(n) = aT(n/b) + \Theta(n^c)$$

1. if $c > log_b(a)$, then $T(n) = \Theta(n^c)$
2. if $c = log_b(a)$, then $T(n) = \Theta(n^c log(n))$
3. if $c < log_b(a)$, then $T(n) = \Theta(n^{log_b(a)})$

merge sort:

```
if n == 1
    return
merge_sort(A[1..n/2])
merge_sort(A[n/2+1..n])
merge(A[1..n/2], A[n/2+1..n])
```

time: $O(nlog(n))$

counting inversions:
just add a counter in merge function
time: $O(nlog(n))$

closest pair:

```
closest_pair(P)
    if |P| <= 3
        brute force
    else
        Q = left half of P
        R = right half of P
        (p1,q1) = closest_pair(Q)
        (p2,q2) = closest_pair(R)
        d = min{dist(p1,q1), dist(p2,q2)}
        delete all points in P that are more than d away from the middle line
        sort remaining points in P by y-coordinate
        scan points in P from top to bottom
            for each point p
                consider only 7 points below p
                compute distance between p and each of the 7 points
                if any distance is less than d
                    update d
```

time: $O(nlog(n))$

Dynamic Programming

optimal matrix mul:
Matrix $M_1, M_2, ..., M_n$ with dimensions $r_1, r_2, ..., r_{n+1}$
$C[i,j] = min_{i<k\le j}(C[i,k-1] + C[k,j] + r_i r_k r_{j+1})$
$C[i,i] = 0$

最优三角剖分:
$C[i,j] = min_{i<k\le j}(C[i,k] + C[k+1,j] + w(v_{i-1}, v_k, v_j))$
$C[i,i] = 0$

0-1 knapsack:
$$V[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ V[i-1,j] & \text{if } w_i > j \\ max(V[i-1,j], V[i-1,j-w_i] + v_i) & \text{otherwise} \end{cases}$$

最长公共子序列:
$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1,j-1] + 1 & \text{if } i,j > 0 \text{ and } x_i = y_j \\ max(C[i-1,j], C[i,j-1]) & \text{if } i,j > 0 \text{ and } x_i \ne y_j \end{cases}$$

hpc:
i give up

整数划分:

$$P(K,m)=\begin{cases}1 & \text{if } K=1 \text{ or } m=1\\ P(K,K) & \text{if } K>m\\ 1+P(K,K-1) & \text{if } K=m\\ P(K,m-1)+P(K-m,m) & \text{otherwise}\end{cases}$$

回溯法

3-Coloring:

```
c[k] = 0 for all k
color(1)
```

```
def color(k)
    for color = 1 to 3
        c[k] = color
        if feasible(k)
            if k == n
                output c[1..n]
            else
                color(k+1)
```

template:

```
v = {}
flag = false
advance(1)
if flag
    output v
else
    print "no solution"
```

```
def advance(k)
    for each x in S
        v[k] = x
        if found_solution(k)
            flag = true and return
        else if feasible(k)
            advance(k+1)
```

branch and bound:

```
initialize upper bound = infinity
root = new node
put root in queue
while queue is not empty
    node = remove node from queue
    if node is a leaf
        update upper bound
    else
        for each child of node
            if child is promising
                put child in queue
```

Randomized Algorithms

throwing needles:

with needle of length l, throw n times, hit m times, then $p = m/n = 2l/\pi d$

随机非重复采样问题:

```
while k < m
    i = random(1,n)
    if i not in S
        S = S union {i}
        k = k + 1
```

quick sort:

```
if l < r
    random select pivot x
    swap A[l] and A[x]
    p = partition(A,l,r)
    quick_sort(A,l,p-1)
    quick_sort(A,p+1,r)
```

```
def partition(A,l,r)
    x = A[l]
    while l < r
        while l < r and A[r] > x
            r = r - 1
        A[l] = A[r]
        while l < r and A[l] <= x
            l = l + 1
        A[r] = A[l]
    A[l] = x
    return l
```

fingerprints:
for n bit binary string x, select a random prime number p,
$I_p(x) = I(x) \mod p$, where $I(x) = \sum_{i=1}^{n} x_i 2^{i-1}$

in case of original string x,
$I_p(x) = I(x) \mod p$, where $I(x) = \sum_{i=1}^{n} x_i s^{i-1}$, s is the length of alphabet

if $I_p(x) \neq I_p(y)$, then $x \neq y$
but if $I_p(x) = I_p(y)$, then $x \neq y$ with probability $1/p$
note: $\pi(x) \approx x/\ln(x)$, if $k < 2^n$

pattern matching with fingerprints:

```
random select prime p
string X with length n
string Y with length m
alphabet size c
W_p = c^m mod p
calculate I_p(Y) and I_p(X[1..m])
for i = 1 to n-m+1
    if I_p(X[i..i+m-1]) = I_p(Y)
        if X[i..i+m-1] = Y
            return i
    I_p(X[i+1..i+m]) = (c * I_p(X[i..i+m-1]) - X[i]W_p + X[i+m]) mod p
return 0
```